NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

VISUAL DATABASE QUERY LANGUAGE

by

Ron Z. Chen

June 1999

Thesis Advisor: Thomas Wu
Second Reader: Chris Eagle

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1999	Master's Thesis

4. TITLE AND SUBTITLE
VISUAL DATABASE QUERY LANGUAGE

5. FUNDING NUMBERS

6. AUTHOR(S)
Chen, Ron Z.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Structure Query Language (SQL) is the most widely used query language in the modern relational database management system (DBMS). Its use is straightforward for simple queries, but it gets complicated, hard to comprehend and express for the complex queries. In term of easy-of-use, Data Flow Query Language (DFQL) represents graphical user interface to the relational model based on a dataflow diagram, and retains all the power of SQL and is equipped with an easy to use facility for extending the language.

With Java's flexibility and power, it is possible to build such system that allows the users to login any relational database through JDBC, graphically view the database structure, and implement the DFQL to query the data from the database.

The design recommendations and implementation of a prototype are the primary research areas of this thesis.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Structure Query, SQL, Data Flow Query Language, DFQL, Java, JDBC, database structure	270
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFI- CATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited

VISUAL DATABASE QUERY LANGUAGE

Ron Z. Chen
Computer Specialist GS-13, Defense Manpower Data Center
B.S., San Jose State University, 1991

Submitted in partial fulfillment of the
requirements for the degree of

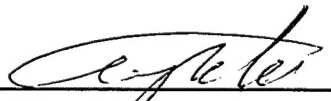
MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

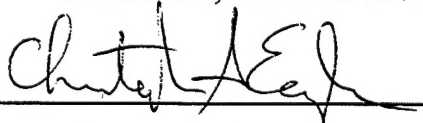
NAVAL POSTGRADUATE SCHOOL
June 1999

Author: 

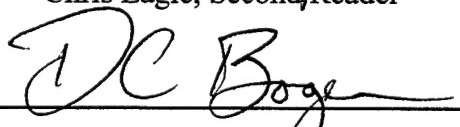
Ron Z. Chen

Approved by: 

Thomas Wu, Thesis Advisor



Chris Eagle, Second Reader



Dan Boger, Chairman
Department of Computer Science

ABSTRACT

Structure Query Language (SQL) is the most widely used query language in the modern relational database management system (DBMS). Its use is straightforward for simple queries, but it gets complicated, hard to comprehend and express for the complex queries. In term of easy-of-use, Data Flow Query Language (DFQL) represents graphical user interface to the relational model based on a dataflow diagram, and retains all the power of SQL and is equipped with an easy to use facility for extending the language.

With Java's flexibility and power, it is possible to build such system that allows the users to login any relational database through JDBC, graphically view the database structure, and implement the DFQL to query the data from the database.

The design recommendations and implementation of a prototype are the primary research areas of this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENT

First, I must thank God for helping me to overcome all the obstacles through the years to achieve this goal. I would like to thank my graduate advisor – Professor Yutaka Kanayama for his kindness and priceless advises on my course matrix. I would also like to thank my thesis advisor Professor Thomas Wu and LCDR Chris Eagle for their patience, programming suggestion, and guidance.

Furthermore, I must thank all of my friends and family whose words of encouragement kept me going at each step of this evolution. Finally, I would like to thank my wife, Haiyan Huang, and my daughter, Karen, for their support and inspiration.

I. INTRODUCTION

A. OVERVIEW

Structured Query Language (SQL) is the most widely used query language in the modern relational database management system (DBMS). Its use is straightforward for simple queries, but it gets complicated, hard to comprehend and express for the complex queries. In terms of easy-of-use, Data Flow Query Language (DFQL) represents graphical user interface to the relational model based on a dataflow diagram, and retains all the power of SQL and is equipped with an easy to use facility for extending the language.

With Java's flexibility and power, it is possible to build such system that allows the users to login any relational databases through JDBC, graphically view the database structure, and implement the DFQL to query the data from the database.

B. STRENGTHS OF SQL

- SQL supports a simple data structure, namely tables [Ref. 1:p. 734].
- SQL supports the relational algebra operators of PROJECT, SELECT, and JOIN and operates on entire relations to generate new relations [Ref. 1:p. 734].
- SQL provides physical data independence to a large extent. Indexes may be added and deleted freely. There is also logical data independence for users in

the sense that they can concentrate on data of interest to them by defining appropriate SQL views [Ref. 1:p. 734].

- SQL combines table creation, querying and updating, and view definition into uniform syntax [Ref. 1:p. 734].
- SQL can be used both as a stand-alone query language and within a general-purpose language by embedding it within a host language [Ref. 1:p. 734].
- The language can be optimized and compiled or may be interpreted and executed on line [Ref. 1:p. 734].

C. WEAKNESSES OF SQL

- Difficulty in comprehending SQL [Ref. 2:p. 3] – SQL is a primarily a declarative query language. Embedding it into a procedural, 3rd generation host programming language compensates the lack of procedure nature of SQL. This allows the user to use the host language to accomplish operations that are difficult or impossible to code in the query language.
- Difficulty in expressing universal quantification [Ref. 2:p. 3] – SQL's lack of a specific "for all" operator forces one to use "negative logic" (the existential quantifier **NOT EXISTS** to achieve the result of universal quantification).
- Lack of orthogonality [Ref. 2:p. 4] – There are numerous examples of arbitrary restrictions, exceptions, and special rules. An example of an unorthogonal construct in SQL is allowing only a single DISTINCT keyword in a SELECT statement contains other nested SELECT's. This increases the

number of special rules to be memorized by the user, decreases its readability, and in general decreases the usability.

- Lack of functional notation [Ref. 2:p. 4] – Complex queries that provide an intermediate result for a higher level query could hide this result from the user through the use of functional notation. This concept is universally adopted in all modern programming languages, but not in SQL.

D. DESCRIPTION OF DFQL

DFQL is a visual relational algebra for the manipulation of relational databases. It can also be extended for object-relational databases, object-oriented databases. It has sufficient expressive power and functionality to allow the user to express database queries. DFQL is relationally complete and includes an implementation of aggregate functions. A facility is provided for the user to create DFQL operators, thus allowing extensibility. DFQL has been developed as a token model graphical dataflow language. The use of the token model implies that each of the defined operators is designed to operator on a stream of tokens over their lifetime. Each Operator will execute once over the life of a given query. The user connecting the desired DFQL operators graphically defines queries. The arguments for the operators flow from the bottom or "output node" of the operator to the top or "input node" of the next operator. Operator execution is initiated by the presence of the requisite input data [Ref. 2:p. 5].

All DFQL operators have the same basic appearance. Each operator has three types of components: the input nodes, the body, and the output node.

One output node may be connected to other operator's input to pass the intermediate result along. The functional paradigm is fully supported by the DFQL notation. The inputs to each operator, or function, arrive at the input nodes of the operator and the result leaves from the output node. All operators of DFQL implement operational closure: output from each operator is always a relation [Ref. 2:p. 5].

II. BACKGROUND

This thesis is inspired by the excellent research paper **"DFQL: Dataflow query language for relational databases"** by Gard J. Clark, C. Thomas Wu, Naval Postgraduate School, Department of Computer Science. The paper is published by The International Journal of Information, Information & Management 27 (1994) 1-15. At the time that this paper published, Gard J. Clark is Lieutenant on active duty in the Navy, with MS degree in Computer Science from the Naval Postgraduate School in 1991. C. Thomas Wu is an Associate Professor of Computer Science at the Naval Postgraduate School in Monterey, California.

DFQL is a new query language, which has been designed to mitigate SQL's ease-of-use problem. DFQL is relational complete, maintains relation operational closure, and is designed to be easily extensible by the end user.

DFQL's advantages accrue from the combination of its visual representation, its dataflow structure, and its operator set. It can be used to express both simple and complex queries in an intuitive manner [Ref. 2:p. 13].

- Power – DFQL is relational complete, and extends first-order predicate logic by the inclusion of grouping operators in both comparison functions and aggregation. The provided set of primitive operators gives the user the capability of coding any desired query.

- Extensibility – The user may extend the DFQL language by coding user-defined operators from the set of provided primitive operators and other already defined user-defined operators.
- Ease-of-use – A dataflow diagram has the capability, especially when using levels of abstraction, to represent even complex problems in an intuitive manner. The ability to form and modify queries incrementally is one of DFQL's most important ease-of-use feature. Since the output of an operator must be a relation, the result of a DFQL operator may always be combined with another DFQL operator to form a more complex query. The combination of all of these features definitely aids the user in the construction of correct queries.
- Visual Interface – The key to implementation of DFQL is the ability for the user to build and modify the DFQL dataflow style queries easily and interactively. DFQL encourages the user to construct queries incrementally, use intermediate results, and take advantage of all the benefits provided by the dataflow approach.

In belief of the great potential of DFQL, it is highly desirable to have a system to implement and extend the DFQL's features and functionalities. This translates the research ideas into a real product.

With the popularity and maturity of the Java® technology from Sun® Microsystem Inc., Java Development Kit 1.2 (JDK 1.2, also name Java 2) is chosen to

implement graphical user interface of the DFQL. The JDK 1.2 is a major upgrade of the Core and Standard Extension APIs of the Java Development Kit. It includes version 1.1 of the Java Foundation Classes (JFC), CORBRA support, a more secure and flexible security model, improvement to the APIs of JDK 1.1, and performance enhancements.

III. ANALYSIS OF JAVA, DATABASE SYSTEMS, AND DFQL

Java, relational database systems, DFQL are combined to make this product come to live. In the following sections, the features and advantages, which are used in this project, of the each components will be analyzed,

A. JAVA

- JFC – Probably the single most important new feature added to JDK1.2 is version 1.1 of the Java Foundation Classes (JFC). JFC is a set of APIs for building the GUI-related components of Java applets and applications [Ref. 3:p. 8]. The APIs included with JFC include the following:
  - The Abstract Windowing Toolkit (AWT) – It provides the capability to create platform-independent, GUI-based programs and is very important contributor to Java's popularity. The AWT of JDK 1.2 has been augmented with many new classes and interfaces that add drawing, printing, and image-processing capabilities, and support the accessibility, Drag and Drop, and Java 2D APIs [Ref. 3:p. 8].
  - Swing – It extends AWT by supplying many more types of GUI components, providing 100% pure Java implementations of these components, and allowing the appearance and behavior of these components to be easily tailored. The new components that are included with Swing include everything from tabbed panes and fancy borders to

sliders and spinners. These new components, in and of themselves, make Swing an outstanding addition to the Java API [Ref. 9:p. 9].

- Java 2D – It provides comprehensive support for two-dimensional drawing, image processing, graphics rendering, color management, and printing. It consists of an imaging model that support line art, text, images, spatial and color transformations, and image composition. The model is device-independent, allowing displayed and printed graphics to be rendered in a consistent manner. The Java 2D API is incorporated into the java.awt and java.awt.image package [Ref. 3:p. 10].

- JDBC – It provides the capability to access databases from Java. JDK 1.2 includes an improved version of the JDBC-ODBC bridge driver and support for JDBC 2.0 [Ref. 3:p. 14].
- Performance – The overall performance of the JDK tools has been greatly improved. First and foremost is the inclusion of a just-in-time (JIT) compiler with JDK. Other performance enhancements include the use of native libraries for some performance-critical Core API classes, improvements to multithreading performance, and reduction in memory usage for string constants [Ref. 3:p. 14].

B. DATABASE SYSTEMS

Two type of relational database management system (RDBMS) are chosen for this product.

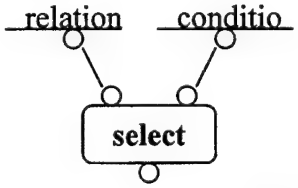
- Oracle Relational Database Management System is the top leading database management system. Its power, flexibility, and high performance make it the first choice for many corporations to host their data. Personal Oracle 7.3.3.0 for Window 95/NT, and Oracle Server 7.3.0 for Windows NT 4.0 are used for testing the product.
- Microsoft Access 97 is the most popular relational database on the computer desktop system. It is used for initial start on the development process. However, since Microsoft uses its proprietary JET® SQL as its Access 97's core query language instead of ANSI SQL, there are many features in ANSI SQL not supported in Access 97. Access 97 can run fine on the simple queries, but will not work on the complex queries. Therefore, Access 97 is dropped from the development.

C. DFQL – DATAFLOW QUERY LANGUAGE

Three broad categories of DFQL operators are defined.

1. Basic DFQL operators – it provides six basic operators derived from the requirements for relational completeness and also the requirement to provide a form of grouping or aggregation [Ref. 2: p. 6].

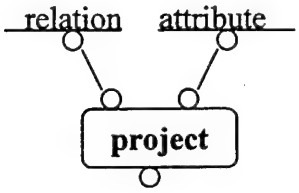
   - Select



Equivalent SQL
SELECT DINSTICT *
FROM relation
WHERE condition;

Figure 1: Select operator

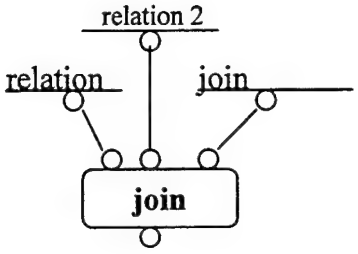
- Project



Equivalent SQL
SELECT DINSTICT attribute list
FROM relation;

Figure 2: Project operator

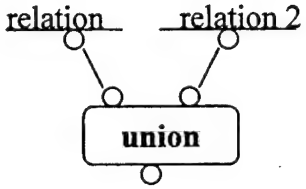
- Join



Equivalent SQL
SELECT DINSTICT *
FROM relation1 r1, relation2 r2
WHERE join condition;

Figure 3: Join operator

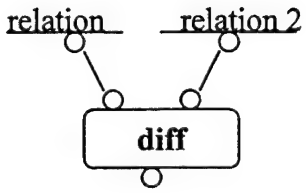
- Union



Equivalent SQL
SELECT DINSTICT *
FROM relation1
UNION
SELECT DINSTICT *
FROM relation2;

Figure 4: Union operator

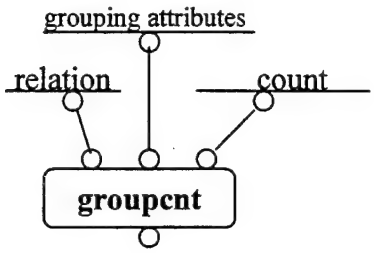
- Diff



Equivalent SQL
SELECT DINSTICT *
FROM relation1
MINUS
SELECT DINSTICT *
FROM relation2;

Figure 5: Diff operator

- Groupcnt

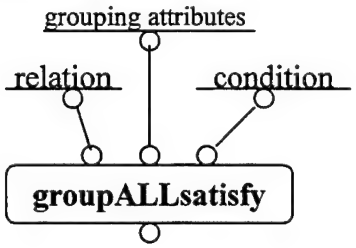


Equivalent SQL
SELECT DINSTICT grouping attributes
COUNT(*) count attributes
FROM relation
GROUP BY grouping attributes;

Figure 6: Groupcnt operator

2. Non-basic primitives (advance) DFQL operators – Most of the additional operators perform operations that are low level. Several could be specified as user-defined operators.

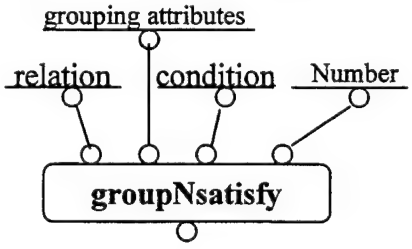
- GroupALLsatisfy



Equivalent SQL
SELECT DINSTICT grouping attributes
FROM relation
WHERE condition
GROUP BY grouping attributes;

Figure 7: GroupALLsatisfy operator

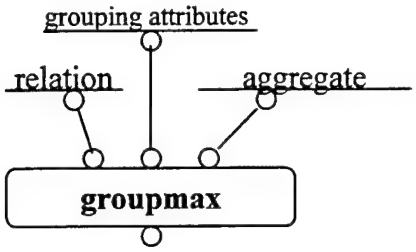
- GroupNsatify – similar to GroupALLsatisfy, but takes an extra argument that allows the user to specify how many of the tuples satisfy the condition for that group to be included in the result. This fourth argument must consist of relational operator (<, >, =, <=, >=, ~=) and a number.



Equivalent SQL
SELECT DINSTICT grouping attributes
FROM relation
WHERE condition and rownum <number>
GROUP BY grouping attributes;

Figure 8: GroupNsatisfy operator

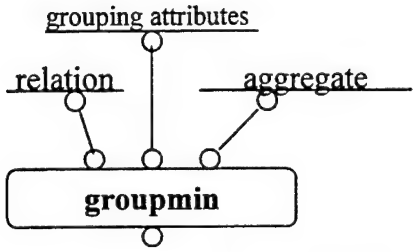
- Groupmax



Equivalent SQL
SELECT DINSTICT grouping attributes
max(aggregate attribute)
FROM relation
GROUP BY grouping attributes;

Figure 9: Groupmax operator

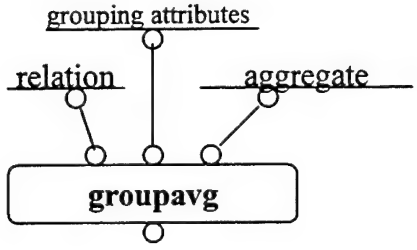
- Groupmin



Equivalent SQL
SELECT DINSTICT grouping attributes
min(aggregate attribute)
FROM relation
GROUP BY grouping attributes;

Figure 10: Groupmin operator

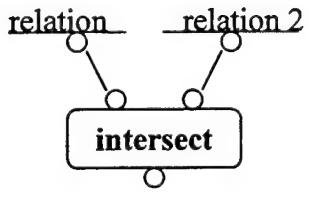
- Groupavg



Equivalent SQL
SELECT DINSTICT grouping attributes
avg(aggregate attribute)
FROM relation
GROUP BY grouping attributes;

Figure 11: Groupavg operator

- Intersect



Equivalent SQL
SELECT DINSTICT *
FROM relation1
intersect
SELECT DINSTICT *
FROM relation2;

Figure 12: Intersect operator

3. User-defined operators – the user can construct operators that look and behave exactly like the primitives. The user can create operators for situations that are unique to his query needs.

IV. PROTOTYPE OF THE GRAPHICAL USER INTERFACE

The system is designed to be simple, intuitive, and yet powerful and flexible. All the complex tasks should not been seen by the user. Therefore, the user interface should contain a few screens as possible for the user to operator.

To fulfill this purpose, there are two major portions of this user interface design.

A. LOGIN SCREEN

This screen allows the user to input the proper information (User name, Password, Database URL, and the Database Driver), in order to log into the target database.

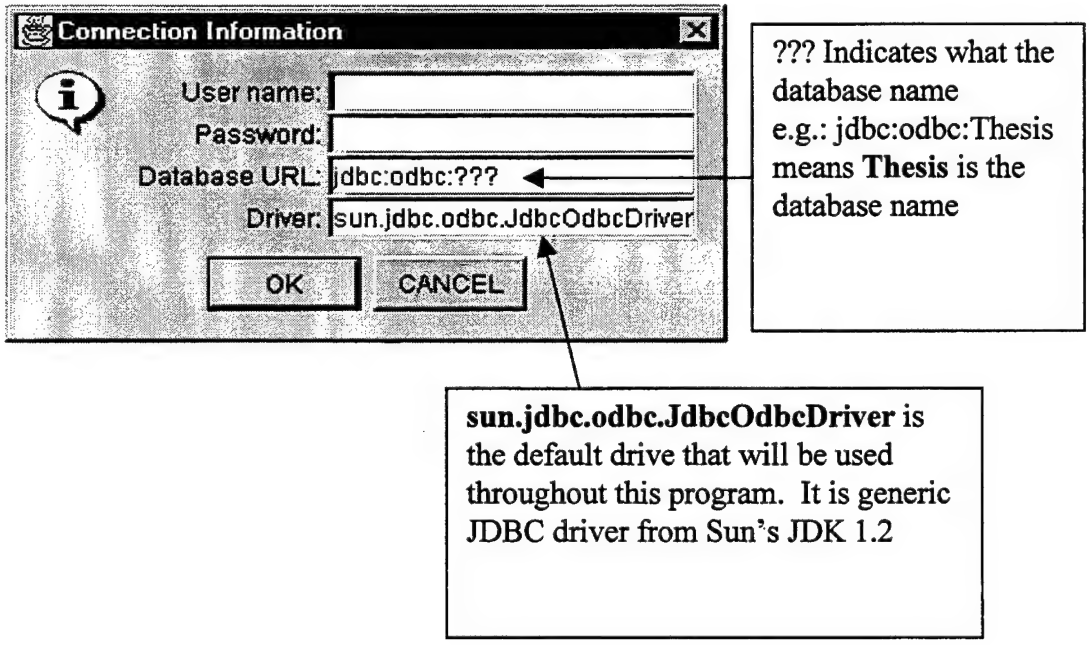


Figure 13: Prototype of login screen

B. MAIN SCREEN

Once the database is connected, the main screen will come up. All the activities will occur on this screen. This screen contains Menu, Toolbar, Split Pane (left pane displays the database metadata, right pane contains another split pane, which runs the regular query, DFQL, and display the result of the query). There is also "About" screen to display what this system is about.

- Main Screen – Menu, Toolbar, Tree panel, Query tab panel, Table Panel

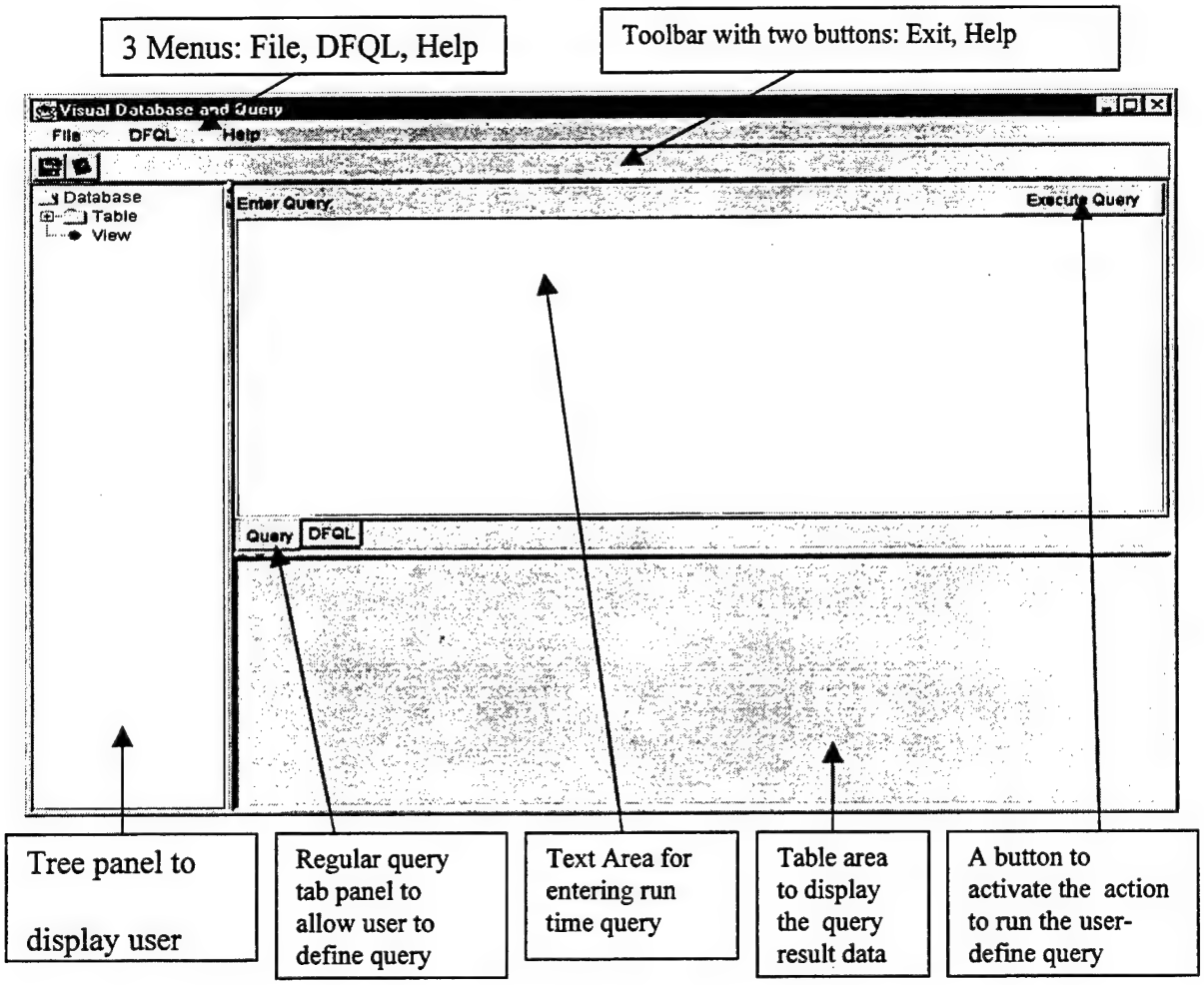


Figure 14. Prototype of main screen

- DFQL tab panel - Basic operator panel (six operators: select, project, join, union, diff, groupcnt).

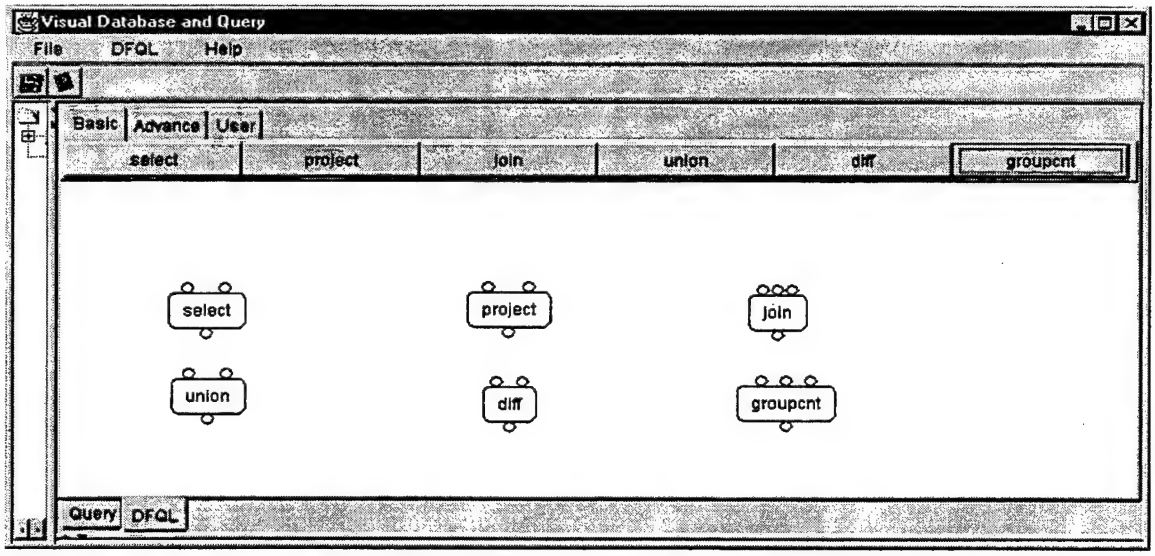


Figure 15: Prototype of basic DFQL operators

- DFQL tab panel - Advance operator panel (six operators: groupALLsatisfy, groupNsatisfy, groupmax, groupmin, groupavg, intersect).

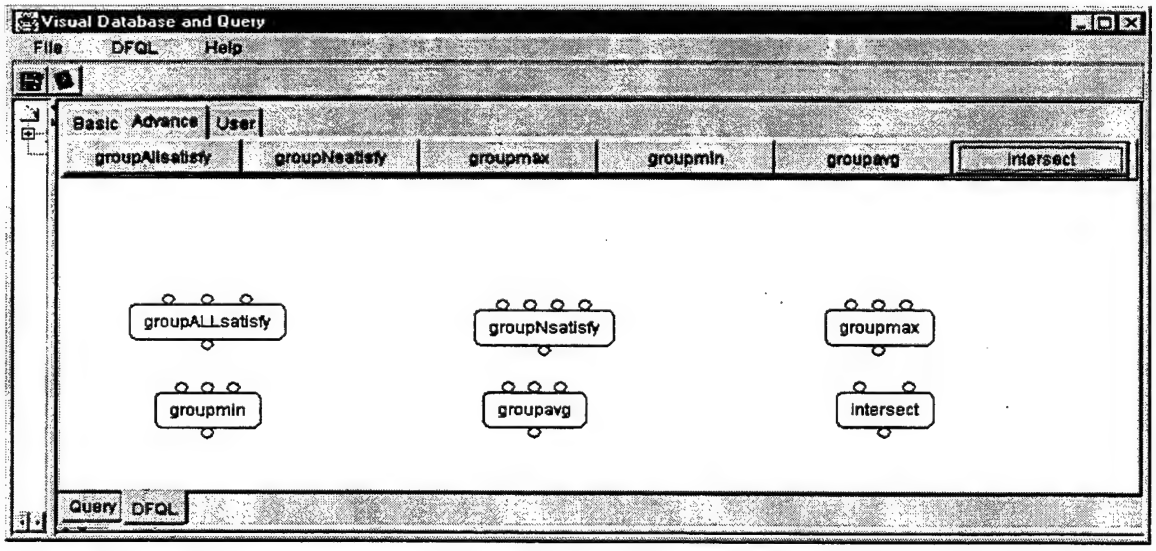


Figure 16: Prototype of advance DFQL operators

DFQL tab panel - User operator panel. Ability to create user operator makes the DQFL very extensible and flexible. However, User Defined Operator is a very complex issue. To make this work, the application must be able to handle the construction mode (as DESIGN mode) and run time mode (as INUSED mode), the information on DESIGN mode must be able to save and use in INUSED mode . This screen shot shows the layout on "User" tab.

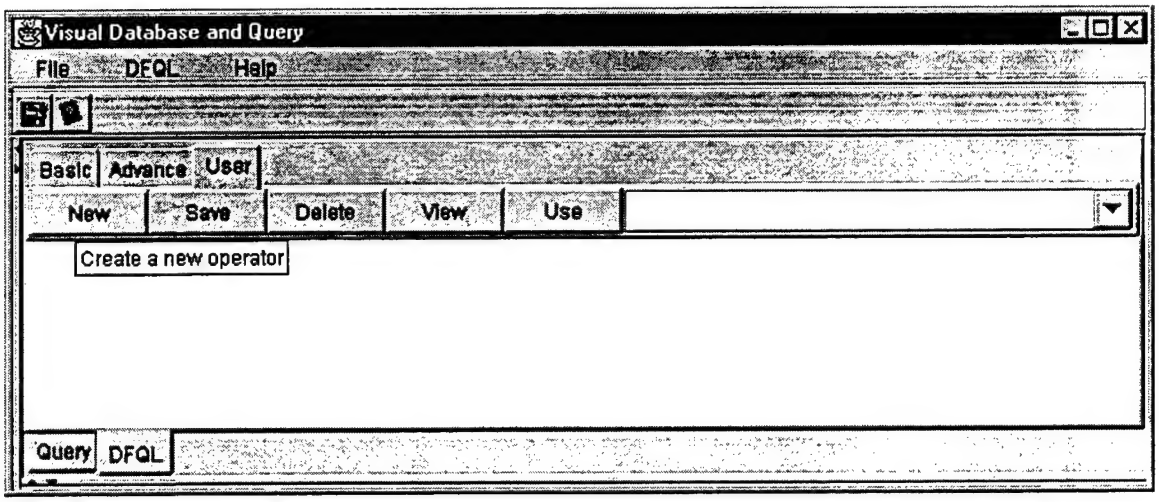


Figure 17. Layout of User Operator Tab

- **New** button allows to create a new user operator
- **Save** button saves the current user defined operator on the canvas into the file with extension **udo,** which stands for **u**ser **d**efined **o**perator.
- **Delete** button deletes the current selected user operator from the ComboBox, and deletes the associated file with this operator from the disk.

- **View** button allows to view the structure of the existing user defined operators by selecting the one from the ComboBox.
- **Use** button allows to use the selected user operator just like the predefined operators to construct the query, run the query, and view the result.
- **ComboBox** lists all the existing user operators which the files is in the working directory. Here is how it works, when the application starts, it scans the working directory to search all the files with extension "udo". All the files with extension "udo" will be put into this combobox.

V. IMPLEMENTATION OF APPLICATION

A. REQUIREMENTS

Hardware: Pentium 233 MHZ, 64 MB RAM, 1.2 GB Hard drive

Platform: Window NT 4.0 Workstation (service pack #4)

Software:

- Sun's JDK 1.2
- UltraEdit-32 – Editor for programming
- Microsoft Access 97
- Personal Oracle 7.3.3.0 for Windows 95/NT 4.0
- Oracle Server 7.3.3.0 for Windows NT Server 4.0
- ODBC driver for MS Access 97 database
- ODBC driver for Oracle 7.3.3.0

B. CONCEPTUAL DESIGN

- User inputs user name and password to login to the target database by specifying the database URL and database Driver). The default database driver will be JDBC ODBC driver from Sun's JDK 1.2.
- . Any databases with ODBC driver proper installed on Windows 9x/NT should be connected by this program.
- Once the database is connected, the program will traverse the database metadata, and show the entire **user** table and view structures.

- Program provides the capabilities for users to execute regular SQL statement, and view the result.
- Most importantly, users can use DFQL operators to develop the data flow diagram graphically, then view the result on the screen. The database flow diagram can be saved for later reused.
- There are 3 levels of DFQL operators: *predefined Basic operators*, *predefined Advance operators*, *user-defined operators* which can be saved and reused.

C. LOGICAL DESIGN

The following diagram shows the general implementation of the program

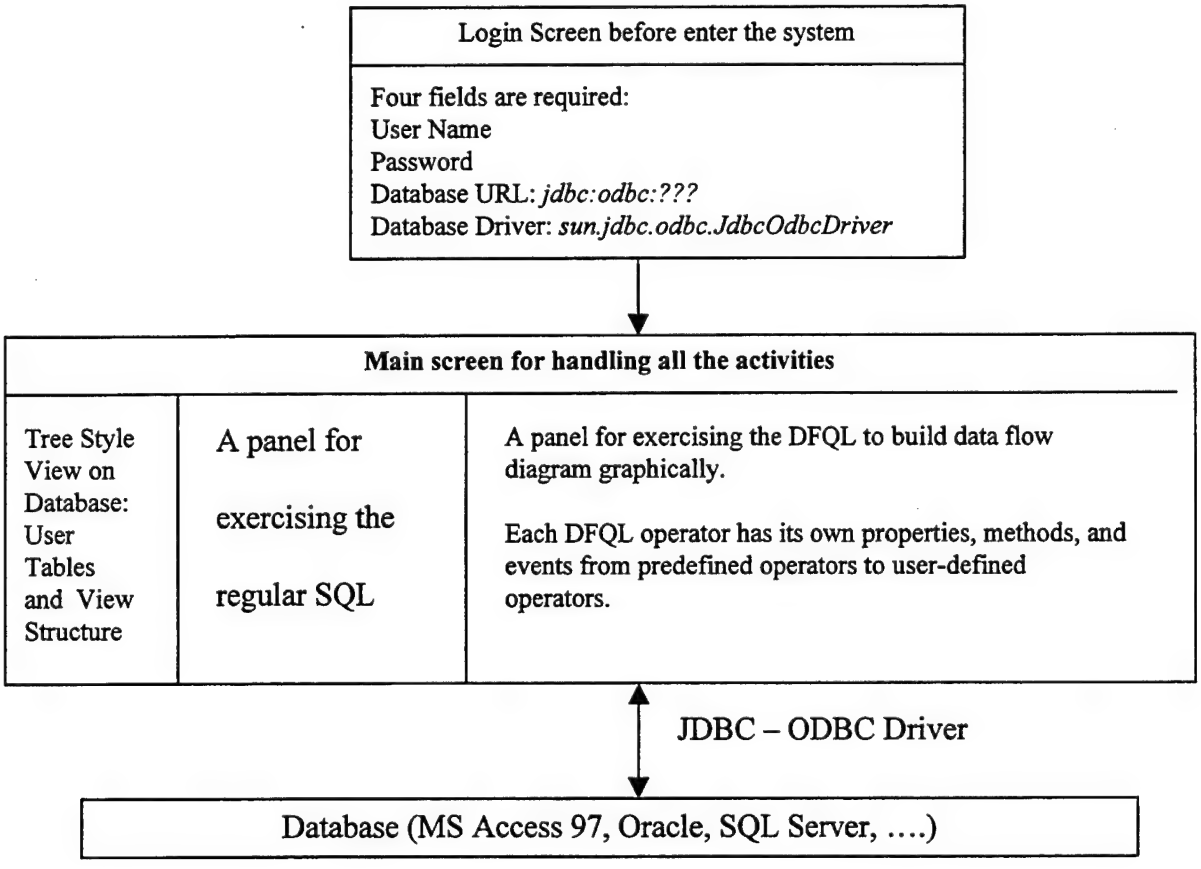


Figure 18: Diagram for logical Design

D. PHYSICAL DESIGN

There are 3 categories API from JDK 1.2 are used extensively in this program: AWT (Graphics2D), Swing Set, Database.

The following Java class files are presented as the hierarchical structure (sort of) of the program.

- *FrameMain.java* - the main frame for the project. This is also the entry point of the program. It contains several classes object: *DB.java*, *TableSorter.java*, *MyTableModel.java*, *DFQL.java*, *LoginDialog.java*, *AboutBox.java.*
- *LoginDialog.java* - A login modal dialog to allow user to input the login information to the database
- *AboutBox.java* – A modal dialog to display the general information what this program is about.
- *DB.java* – A wrapper class for handling all the database activities. Connect to the database, traverse the database, obtain metadata of the table and view structure, execute user-defined queries.
    - *ToolTipTree.java* - This is the extended JTree class with the capability to display the ToolTip text for the tree node.
    - *MultiLineToolTipUI.java* – A extended JToolTip class for handling multiline ToolTip User Interface. The metadata of the database on the tree panel will display on this multiline Tooltip object.

- *TreeNodeName.java* - This is the class for holding the name of the tree node and tooltip text for this tree node.

- *DFQL.java* - A window class for handling DFQL panel. This class contains the *DFQLCanvas.java*, and all the DFQL operators that extend from the base class *Operator.java.* The are four inner classes as utility class inside this class.

    1. *FileType* – This inner class is for file filter base on the file type by implementing the interface **FilenameFilter**. This is particular for OperatorUser class to save/open the user defined operator file into/from the disk.
    2. *ListenerBasicOperators* – This inner class is for handling all the basic operators' actions by implementing the interface ActionListener.
    3. *ListenerAdvanceOperators* – This inner class is for handling all the advance operators' actions by implementing the interface ActionListener.
    4. *ListenerUserDefinedAction* – This inner class is for handling all the user defined operators' actions by implementing the interface ActionListener. However, this class is a lot more complex than *ListenerBasicOperators* and *ListenerAdvanceOperators*, because it has to deal with two total different situation: DESIGN and INUSED mode, for the user defined operator.

- *DFQLCanvas.java* - A window class for DFQL Canvas for handling the graphical operators and inter-relation on each other, and the mouse motion movement on this canvas.
    - *PropertyWindow.java* - A extended dialog class for display operator's property.
- *Operator.java* - A base class for variety of DFQL operators. It defines the common properties, methods, events, which can be extended for particular DFQL operators. Such as implements the MouseListener, MouseMotionListener interface for capturing the mouse movement, Externalizable interface for serializing the operator object for reading and writing.
    - *OperatorSelect.java* – This is the class that extends from class Operator.java. It performs the **select** operation.
    - *OperatorProject.java* - This is the class that extends from class Operator.java. It perfroms the **project** operation.
    - *OperatorJoin.java* - This is the class that extends from class Operator.java. It performs the **join** operation.
    - *OperatorUnion.java* - This is the class that extends from class Operator.java. It performs the **union** operation.
    - *OperatorDiff.java* - This is the class that extends from class Operator.java. It performs the **diff** operation.

- *OperatorGroupcnt.java* - This is the class that extends from class Operator.java. It performs the **project** operation.
- *OperatorGroupALLsatisfy.java* - This is the class that extends from class Operator.java. It performs the **GroupALLsatisfy** operation.
- *OperatorGroupNsatisfy.java* - This is the class that extends from class Operator.java. It performs the **GroupNsatisfy** operation.
- *OperatorGroupmax.java* - This is the class that extends from class Operator.java. It performs the **Groupmax** operation.
- *OperatorGroupmin.java* - This is the class that extends from class Operator.java. It performs the **Groupmin** operation.
- *OperatorGroupavg.java* - This is the class that extends from class Operator.java. It performs the **Groupavg** operation.
- *OperatorIntersect.java* - This is the class that extends from class Operator.java. It performs the **Intersect** operation.
- *OperatorUser.java* – This is the class that extends from class Operator.java, but uses for user defined operator. This is a very complex and long class. There are two major situations that must be paid close attention. 1. **DESIGN mode** - user wants to define a DFQL operator base on the available operators (basic operators, advance operators, and exist user operators). 2. **INUSED mode** -

user uses the current user defined DFQL operator to construct the query.

- *InputBarNode.java* - This is the class that uses as helper class to record the input node information for the OperatorUser class.

- *TableSorter.java* – This class allows the Table can be sorted by clicking the column name.

  This class is extended from class *TableMap.java*

  - *TableMap.java* – This is the class that implements TableModelListener to provides most of common table behaviors.

- *MyTableModel.java* - A class displays the data to the table format (rows and columns)
- *ExampleFileFilter.java* – An extended FileFilter class for allowing FileChoose class object to select the default file type to be open or saved.

E. EXECUTION OF APPLICATION

1. Setup ODBC connection

The database that used for this testing is created at Personal Oracle 7.3.3.0 for Windows 95/NT 4.0. Before running this application, the ODBC should be setup properly on the Windows NT platform. Here is the step to setup ODBC:

- From "Control Panel", double click "ODBC" icon to activate the ODBC setup.

- Select System DSN tab (note: DSN means Data Source Name, The User DSN means these will be specific to the user who has logged in on the machine. System DSN means the these will appear if any user is logged in. Therefore, it is better to create System DSN).
- Click the Add button.
- There is a list of installed ODBC drivers.
- Select Oracle73 and click Finish button.
- In the "**Oracle7 ODBC Setup**" dialog, enter "**csthesis**" **on "Data Source Name**" section, "NPS CS Thesis" on "**Description**" section, "**2:**" on "**SQL*Net Connect String**" section (**2:** means that the Oracle database is resided on the local machine, for remote Oracle database, run "SQL Net Easy Configuration" program to establish the connection).
- Now, the DSN is set and ready to run the Java program by JDBC-ODBC driver.

2. Login into the database

Four elements are required in order to connect the database.

User name: npscs

Passowrd: npscs

Database name: csthesis

Driver: sun.jdbc.odbc.JdbcOdbcDriver

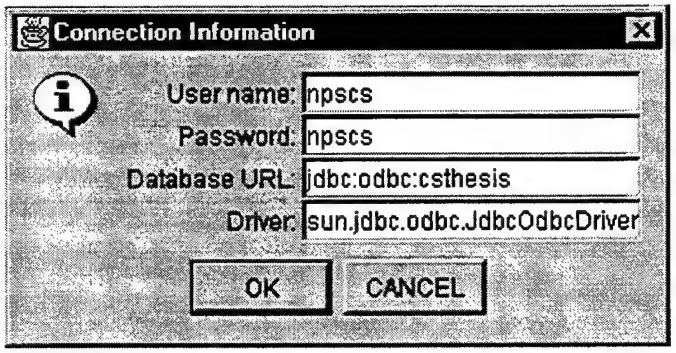


Figure 19: Execution of login process

3. View of the database structure

The database metadata will be shown on the multiline tooltip window when the mouse moves to the certain item. This gives the user a general idea what the database structure is.

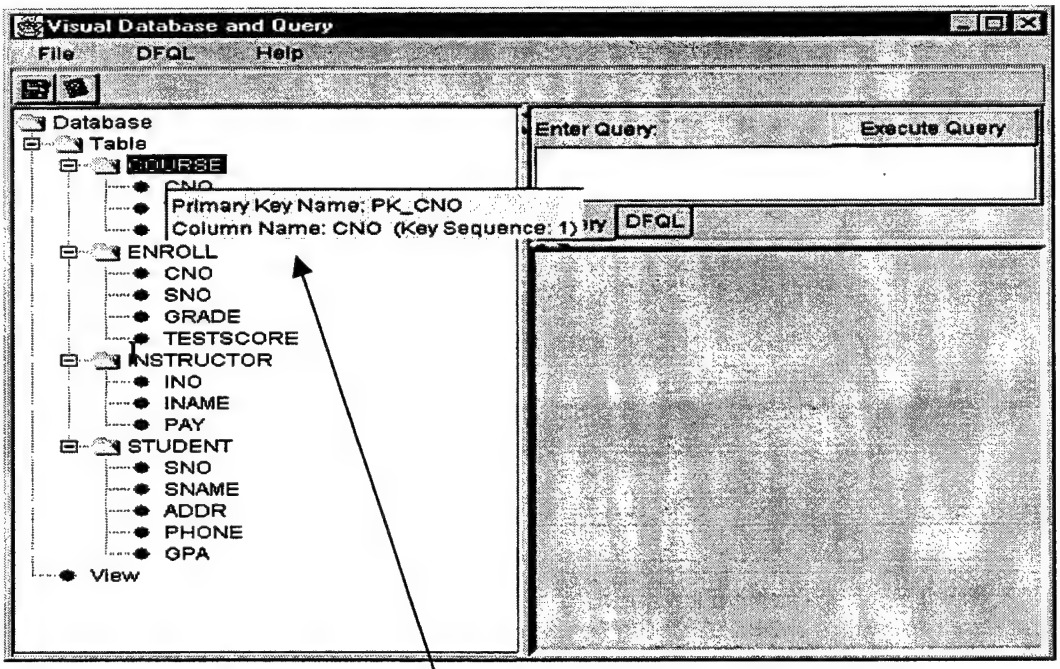


Figure 20. View of Database Structure – Primary Key

The next screen shows the column information on the tooltip window.

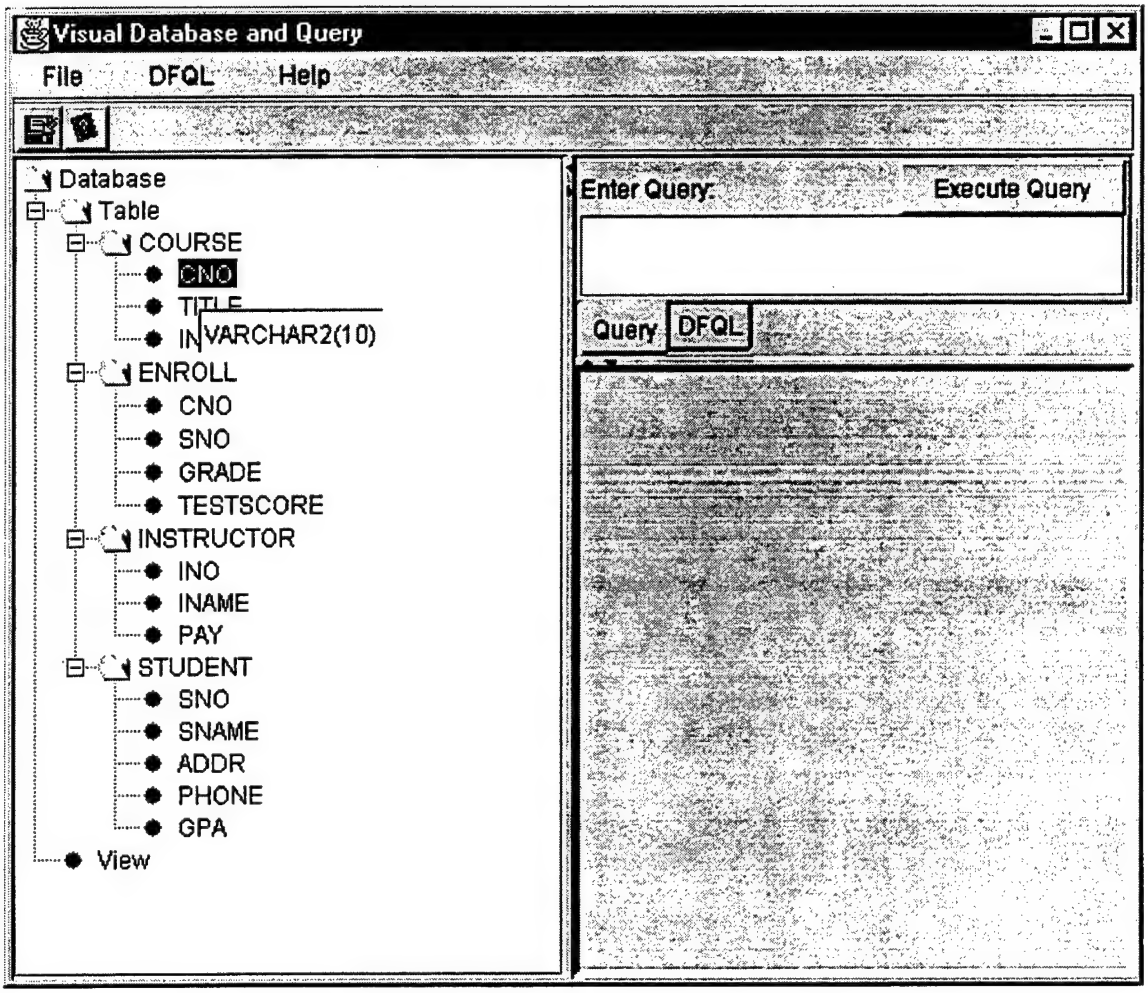
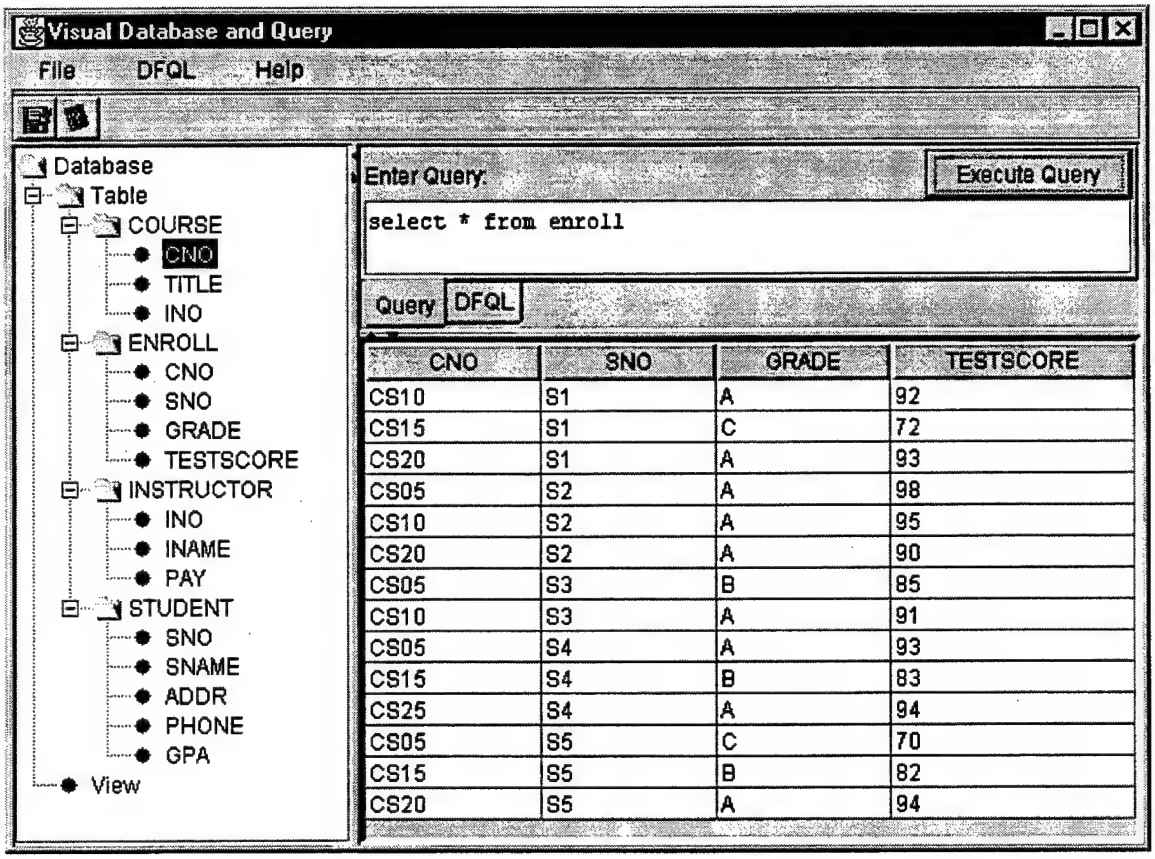


Figure 21. View of Database Structure – Column Information

4. Execute the regular query

On this example, enter the regular query as the following "**select * from enroll**" on the text area, then click "**Execute Query**" button to run this query. The result will show up on the table. User also can click each column header to sort the data on the table.



CNO	SNO	GRADE	TESTSCORE
CS10	S1	A	92
CS15	S1	C	72
CS20	S1	A	93
CS05	S2	A	98
CS10	S2	A	95
CS20	S2	A	90
CS05	S3	B	85
CS10	S3	A	91
CS05	S4	A	93
CS15	S4	B	83
CS25	S4	A	94
CS05	S5	C	70
CS15	S5	B	82
CS20	S5	A	94

Figure 22. Execution of the regular user query

5. Execution of DFQL operator – Select

On this example, first click "**DFQL**" tab to move to the DFQL panel, from "**Basic**" tab, click "**select**" button. "**select**" operator shows up on the canvas. This symbol can be dragged and dropped anywhere inside the canvas. Right-click on the symbol, the property window pops up. Enter "student" on "Relation" field, "gpa > 3.5" on "Condition" field. Click "**OK**" button to accept the input. From **DFQL** menu, select "**Run**" to execute this query. The result shows on the table format.

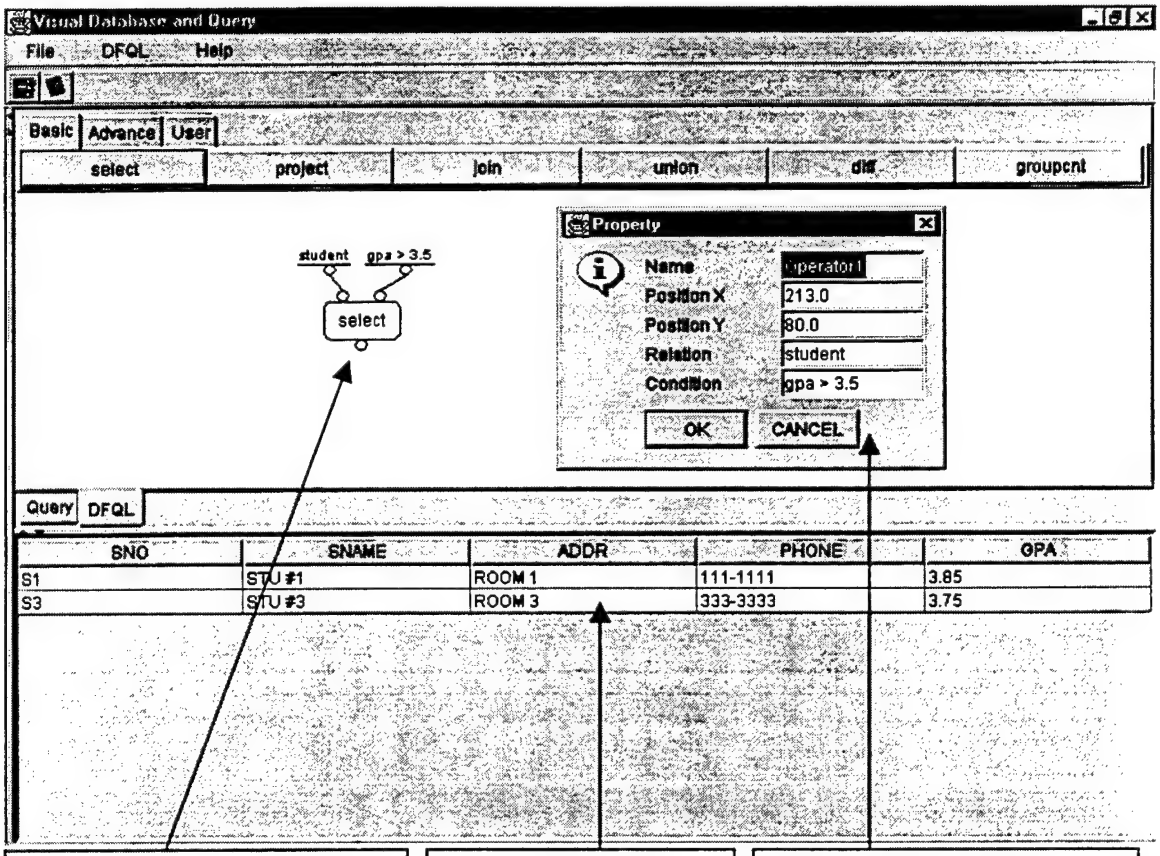


Figure 23. Execution of DFQL operator – Select

6. Execution of DFQL operator – Project

On this example, first click "**DFQL**" tab to move to the DFQL panel, from "**Basic**" tab, click "**project**" button. "**project**" operator shows up on the canvas. This symbol can be dragged and dropped anywhere inside the canvas. Right-click on the symbol, the property window pops up. Enter "enroll" on "Relation" field, "testscore" on "Attribute List" field. Click "**OK**" button to accept the input. From **DFQL** menu, select "**Run**" to execute this query. The result shows on the table format.

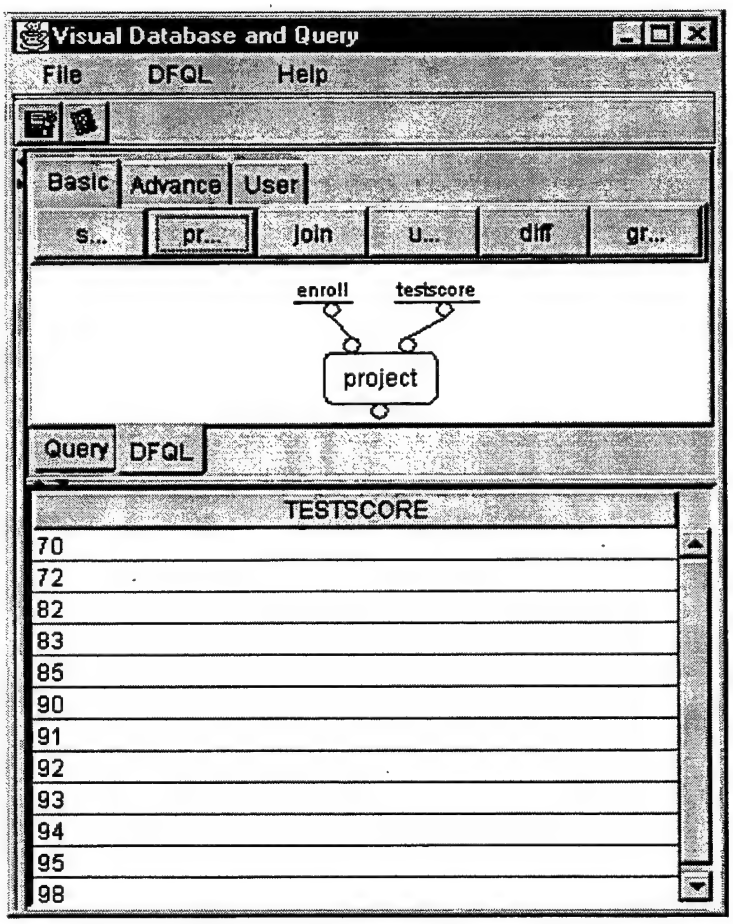
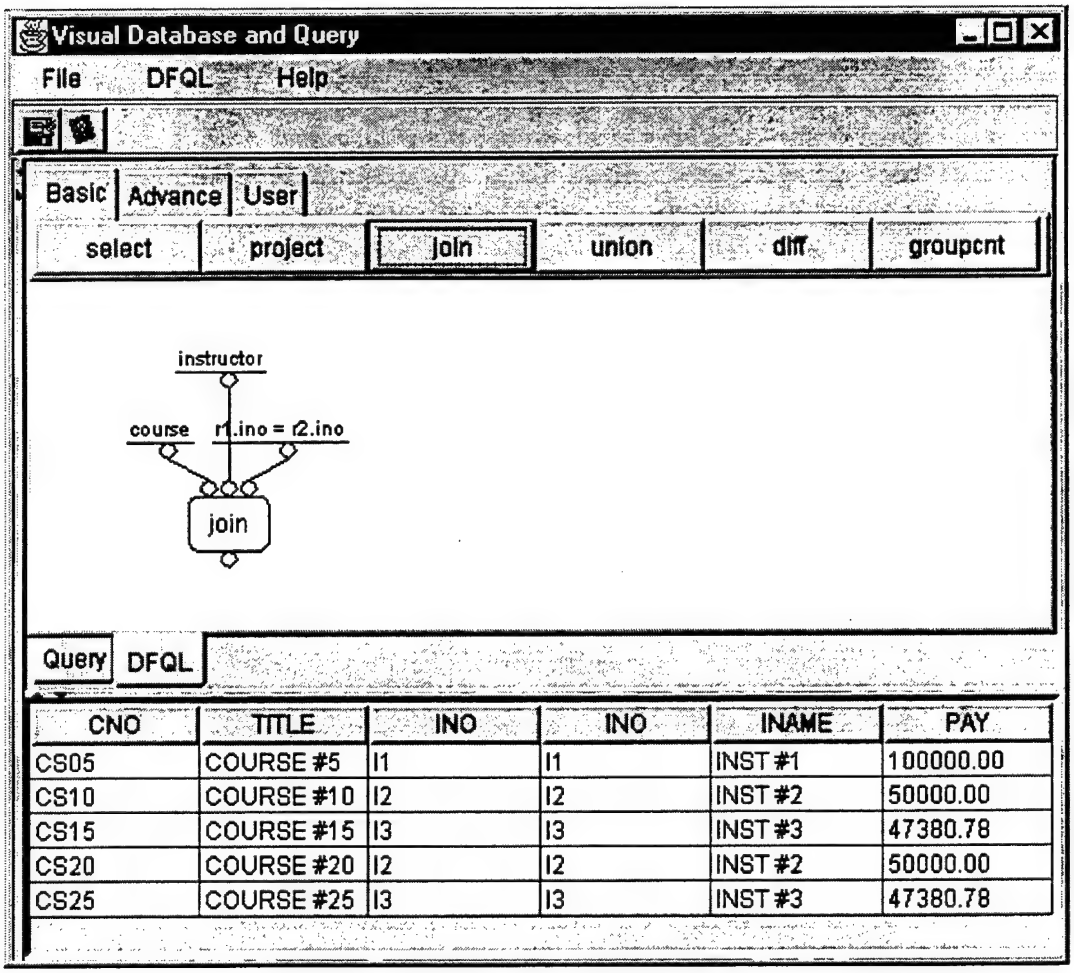


Figure 24. Execution of DFQL operator – Project

7. Execution of DFQL operator – Join

On this example, first click "**DFQL**" tab to move to the DFQL panel, from "**Basic**" tab, click "**join**" button. "**join**" operator shows up on the canvas. This symbol can be dragged and dropped anywhere inside the canvas. Right-click on the symbol, the property window pops up. Enter "course" on "Relation 1" field, "instructor" on "Relation 2" field, "r1.ino = r2.ino" on "Condition" field. Click "**OK**" button to accept the input. From **DFQL** menu, select "**Run**" to execute this query. The result shows on the table format.



CNO	TITLE	INO	INO	INAME	PAY
CS05	COURSE #5	I1	I1	INST #1	100000.00
CS10	COURSE #10	I2	I2	INST #2	50000.00
CS15	COURSE #15	I3	I3	INST #3	47380.78
CS20	COURSE #20	I2	I2	INST #2	50000.00
CS25	COURSE #25	I3	I3	INST #3	47380.78

Figure 25. Execution of DFQL operator – Join

8. Execution of DFQL operator – Union

On this example, first click "**DFQL**" tab to move to the DFQL panel, from "**Basic**" tab, click "**project**" button, drag it to one place, then click "**project**" again, and drag it to another place. Set up each "project" operator's property. Then, click "**union**" button. "**union**" operator shows up on the canvas. Right-click on the symbol, the property window pops up. Enter "Operator1" (or whatever the name of the first operator) on "Relation 1" field, "Operator2" (or whatever the name of the second operator) on "Relation 2" field. Click "**OK**" button to accept the input. From **DFQL** menu, select "**Run**" to execute this query. The result shows on the table format.

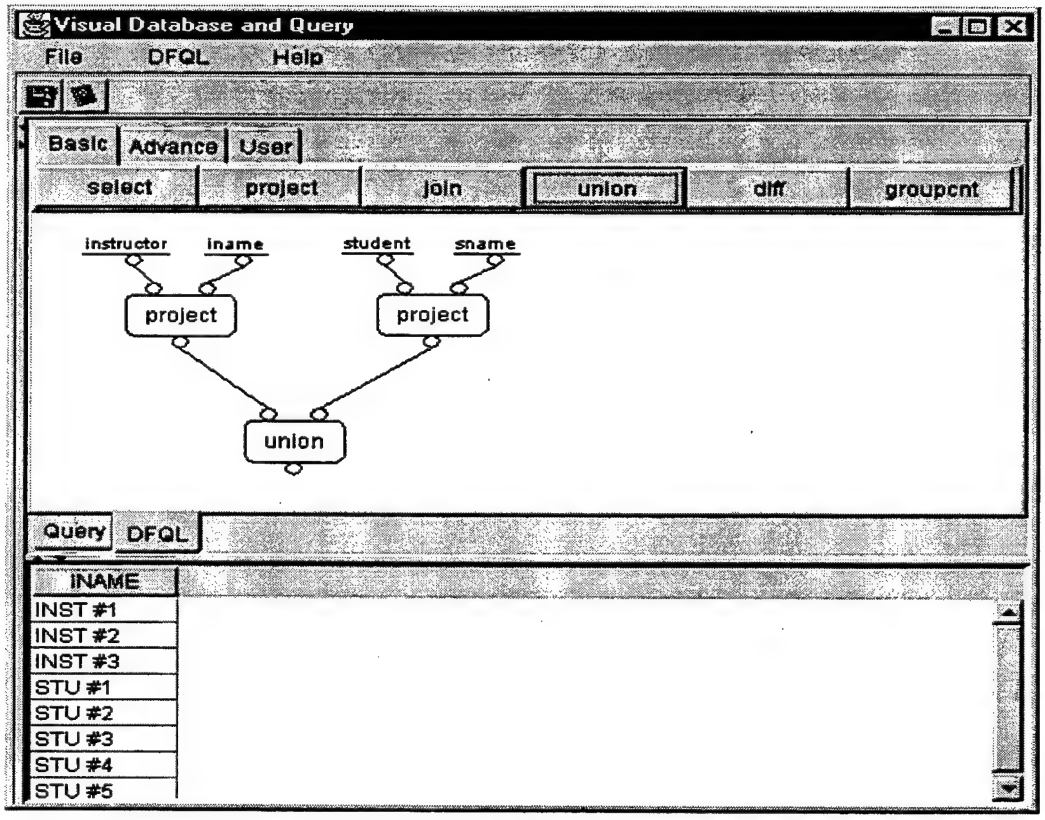


Figure 26. Execution of DFQL operator – Union

9. Execution of DFQL operator – Diff

On this example, first click "**DFQL**" tab to move to the DFQL panel, from "**Basic**" tab, click "**project**" button, drag it to one place, then click "**project**" again, and drag it to another place. Set up each "project" operator's property. Then, click "**diff**" button. "**diff**" operator shows up on the canvas. Right-click on the symbol, the property window pops up. Enter "Operator1" (or whatever the name of the first operator) on "Relation 1" field, "Operator2" (or whatever the name of the second operator) on "Relation 2" field. Click "**OK**" button to accept the input. From **DFQL** menu, select "**Run**" to execute this query. The result shows on the table format.

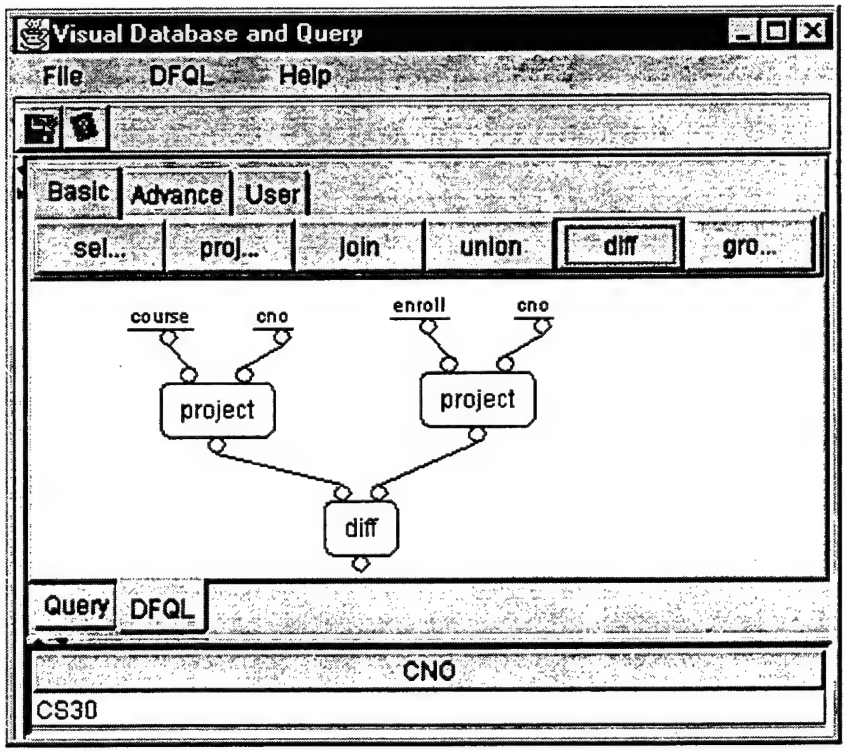


Figure 27. Execution of DFQL operator – Diff

10. Execution of DFQL operator – Groupcnt

On this example, first click "**DFQL**" tab to move to the DFQL panel, from "**Basic**" tab, click "**groupcnt**" button. "**groupcnt**" operator shows up on the canvas. Right-click on the symbol, the property window pops up. Enter "enroll" on "Relation" field, "cno" on "Grouping Attributes" field, "numstudents" on "Count Attribute" field. Click "**OK**" button to accept the input. From **DFQL** menu, select "**Run**" to execute this query. The result shows on the table format.

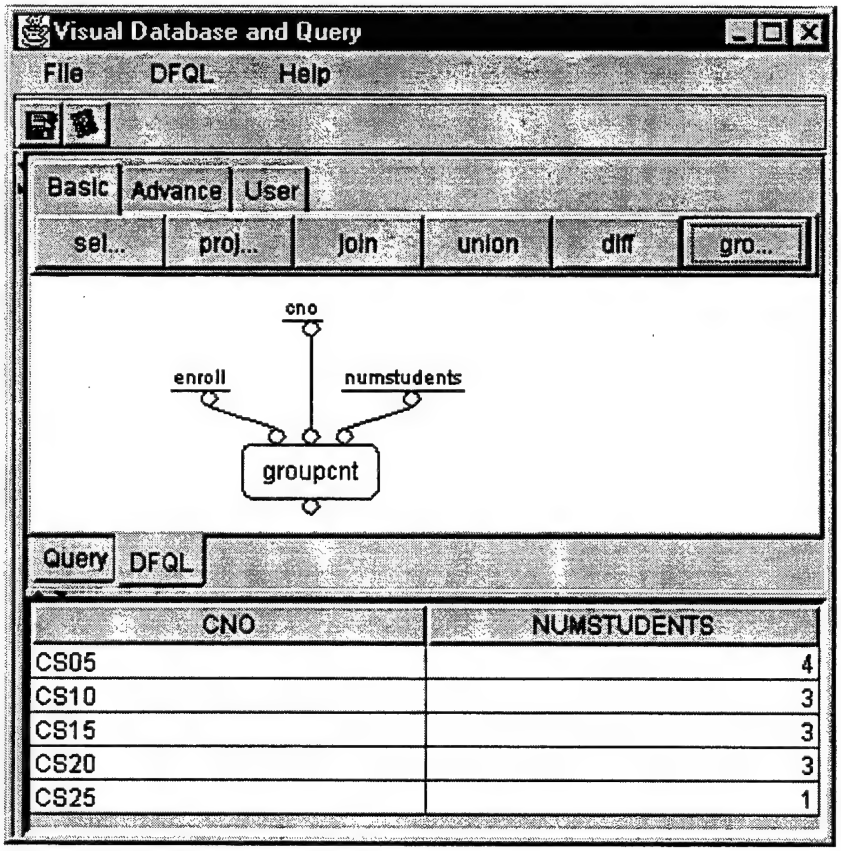
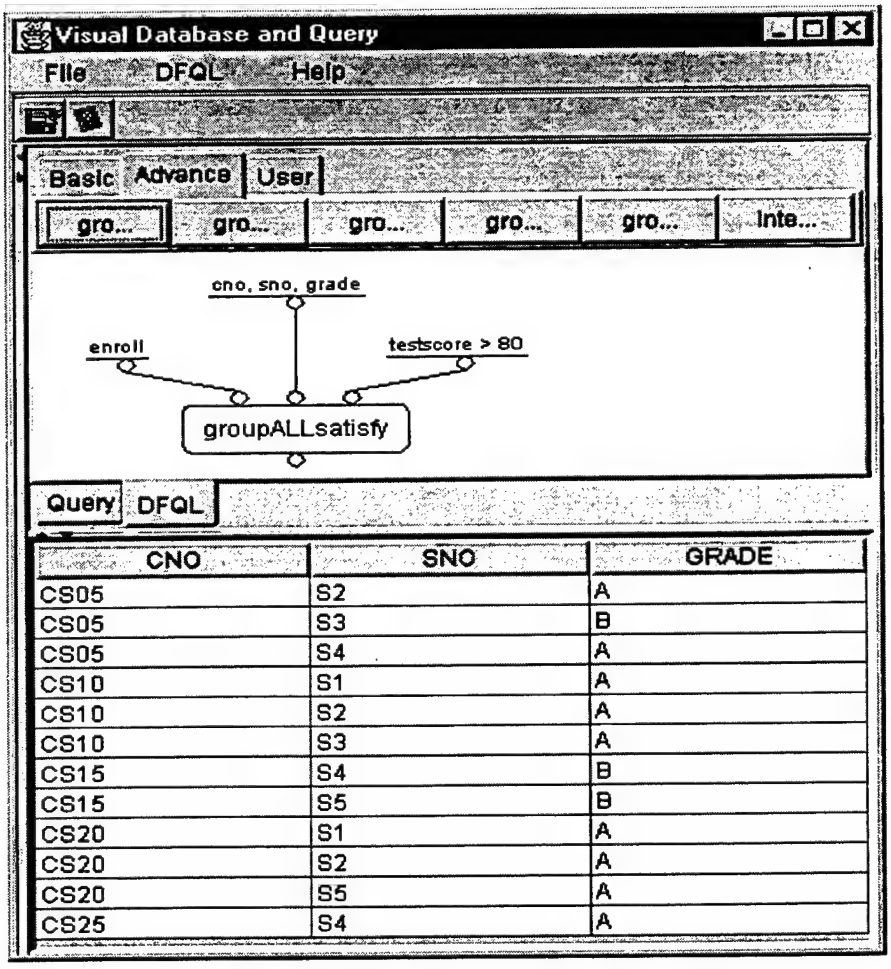


Figure 28. Execution of DFQL operator – Groupcnt

11. Execution of DFQL operator – GroupALLsatisfy

On this example, first click "**DFQL**" tab to move to the DFQL panel, from "**Advance**" tab, click "**groupALLsatisfy**" button. "**groupALLsatisfy**" operator shows up on the canvas. Right-click on the symbol, the property window pops up. Enter "enroll" on "Relation" field, "cno, sno, grade" on "Grouping Attributes" field, "testscore > 80" on "Condition" field. Click "**OK**" button to accept the input. From **DFQL** menu, select "**Run**" to execute this query. The result shows on the table format.



CNO	SNO	GRADE
CS05	S2	A
CS05	S3	B
CS05	S4	A
CS10	S1	A
CS10	S2	A
CS10	S3	A
CS15	S4	B
CS15	S5	B
CS20	S1	A
CS20	S2	A
CS20	S5	A
CS25	S4	A

Figure 29. Execution of DFQL operator – GroupALLsatisfy

12. Execution of DFQL operator – GroupNsatisfy

On this example, first click "**DFQL**" tab to move to the DFQL panel, from "**Advance**" tab, click "**groupNsatisfy**" button. "**groupNsatisfy**" operator shows up on the canvas. Right-click on the symbol, the property window pops up. Enter "enroll" on "Relation" field, "cno, sno, grade" on "Grouping Attributes" field, "testscore > 80" on "Condition" field, "<=4" on "number" field. Click "**OK**" button to accept the input. From **DFQL** menu, select "**Run**" to execute this query. The result shows on the table format.

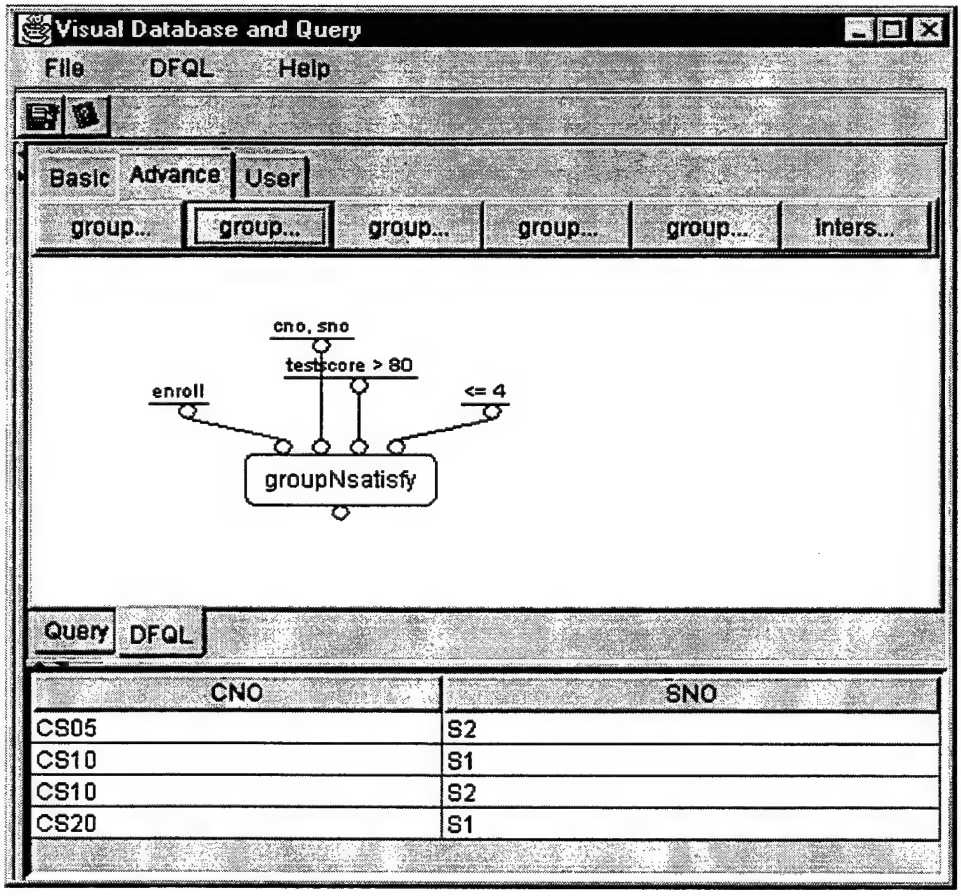


Figure 30. Execution of DFQL operator – GroupNsatisfy

13. Execution of DFQL operator – Groupmax

On this example, first click "**DFQL**" tab to move to the DFQL panel, from "**Advance**" tab, click "**groupmax**" button. "**groupmax**" operator shows up on the canvas. Right-click on the symbol, the property window pops up. Enter "enroll" on "Relation" field, "cno" on "Grouping Attributes" field, "testscore" on "Aggregate Attribute" field. Click "**OK**" button to accept the input. From **DFQL** menu, select "**Run**" to execute this query. The result shows on the table format.

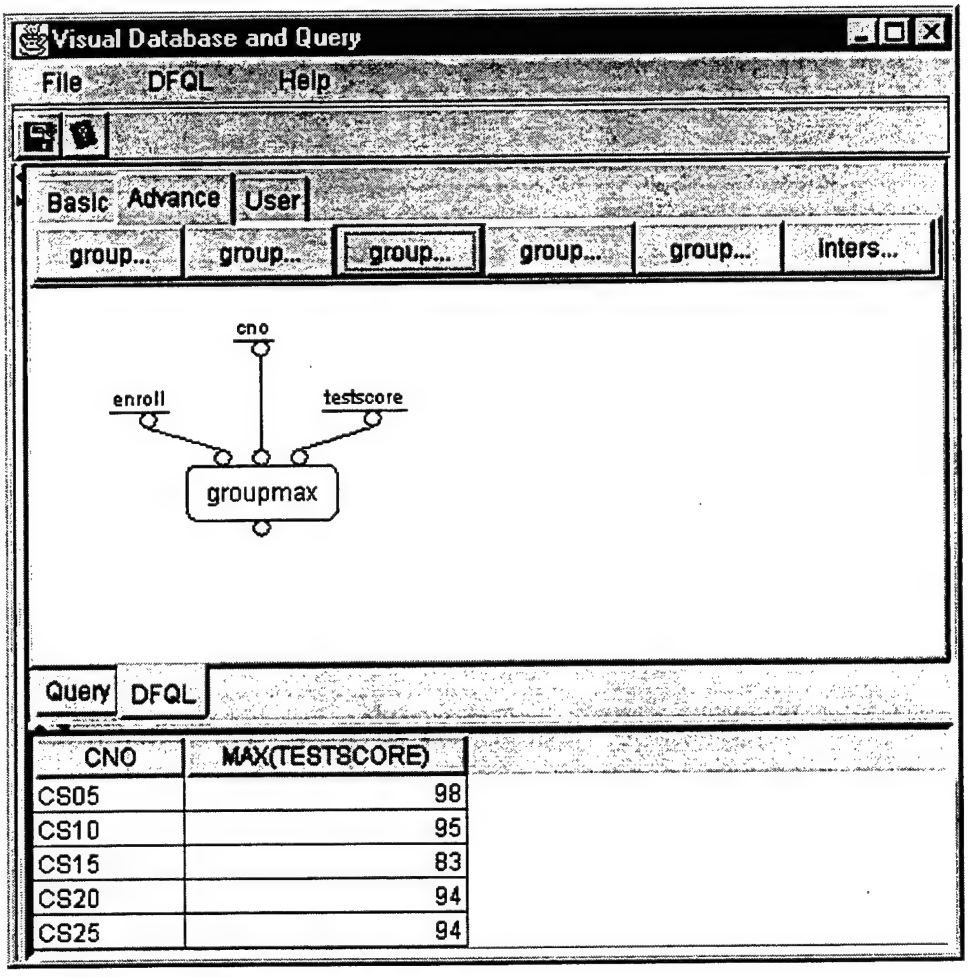


Figure 31. Execution of DFQL operator – Groupmax

14. Execution of DFQL operator – Groupmin

On this example, first click "**DFQL**" tab to move to the DFQL panel, from "**Advance**" tab, click "**groupmax**" button. "**groupmax**" operator shows up on the canvas. Right-click on the symbol, the property window pops up. Enter "enroll" on "Relation" field, "cno" on "Grouping Attributes" field, "testscore" on "Aggregate Attribute" field. Click "**OK**" button to accept the input. From **DFQL** menu, select "**Run**" to execute this query. The result shows on the table format.

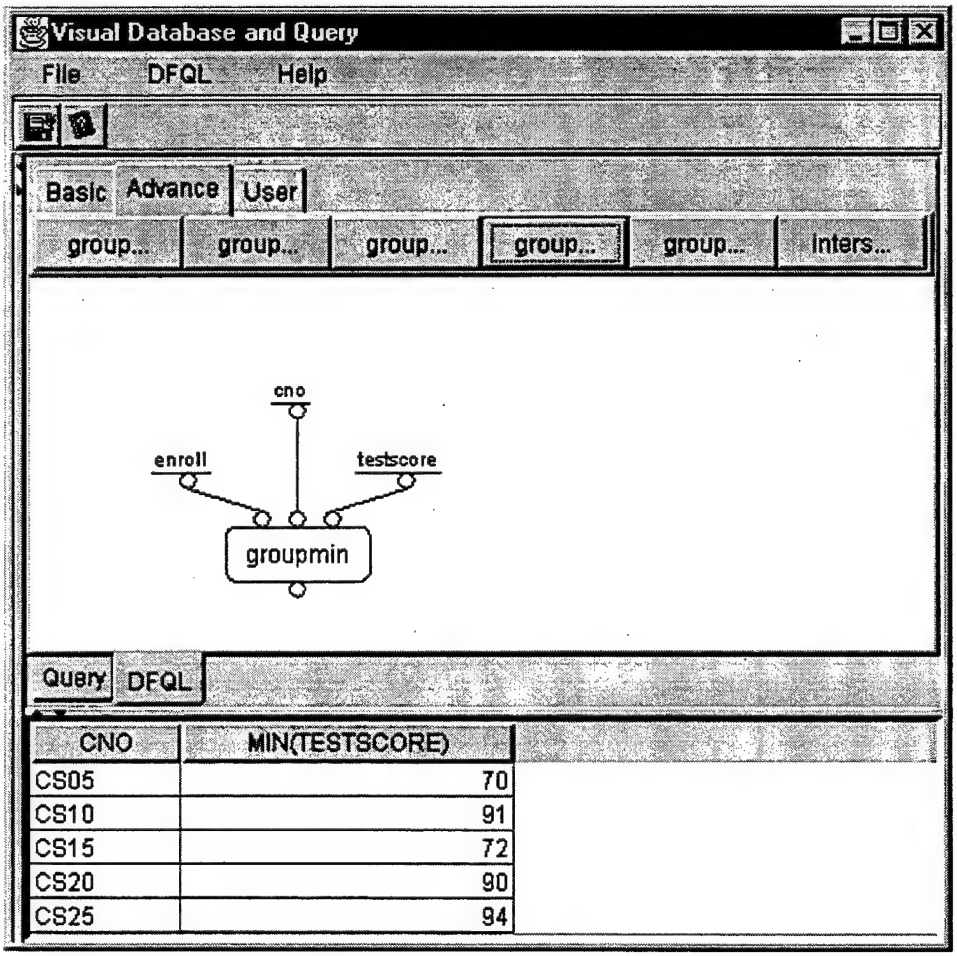
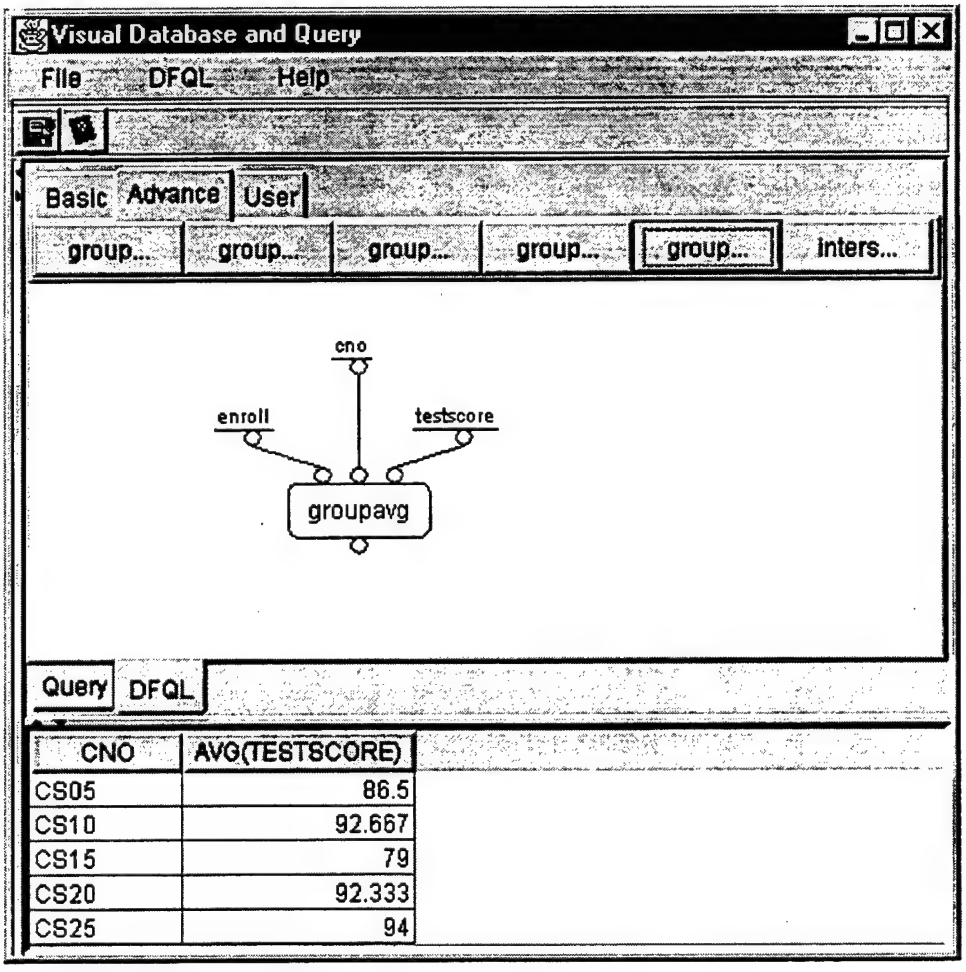


Figure 32. Execution of DFQL operator – Groupmin

15. Execution of DFQL operator – Groupavg

On this example, first click "**DFQL**" tab to move to the DFQL panel, from "**Advance**" tab, click "**groupmax**" button. "**groupmax**" operator shows up on the canvas. Right-click on the symbol, the property window pops up. Enter "enroll" on "Relation" field, "cno" on "Grouping Attributes" field, "testscore" on "Aggregate Attribute" field. Click "**OK**" button to accept the input. From **DFQL** menu, select "**Run**" to execute this query. The result shows on the table format.



CNO	AVG(TESTSCORE)
CS05	86.5
CS10	92.667
CS15	79
CS20	92.333
CS25	94

Figure 33. Execution of DFQL operator – Groupavg

16. Execution of DFQL operator – Intersect

On this example, first click "**DFQL**" tab to move to the DFQL panel, from "**Advance**" tab, click "**select**" button, drag it to one place, then click "**select**" again, and drag it to another place. Set up each "project" operator's property. Then, click "**intersect**" button. "**intersect**" operator shows up on the canvas. Right-click on the symbol, the property window pops up. Enter "Operator1" (or whatever the name of the first operator) on "Relation 1" field, "Operator2" (or whatever the name of the second operator) on "Relation 2" field. Click "**OK**" button to accept the input. From **DFQL** menu, select "**Run**" to execute this query. The result shows on the table format.

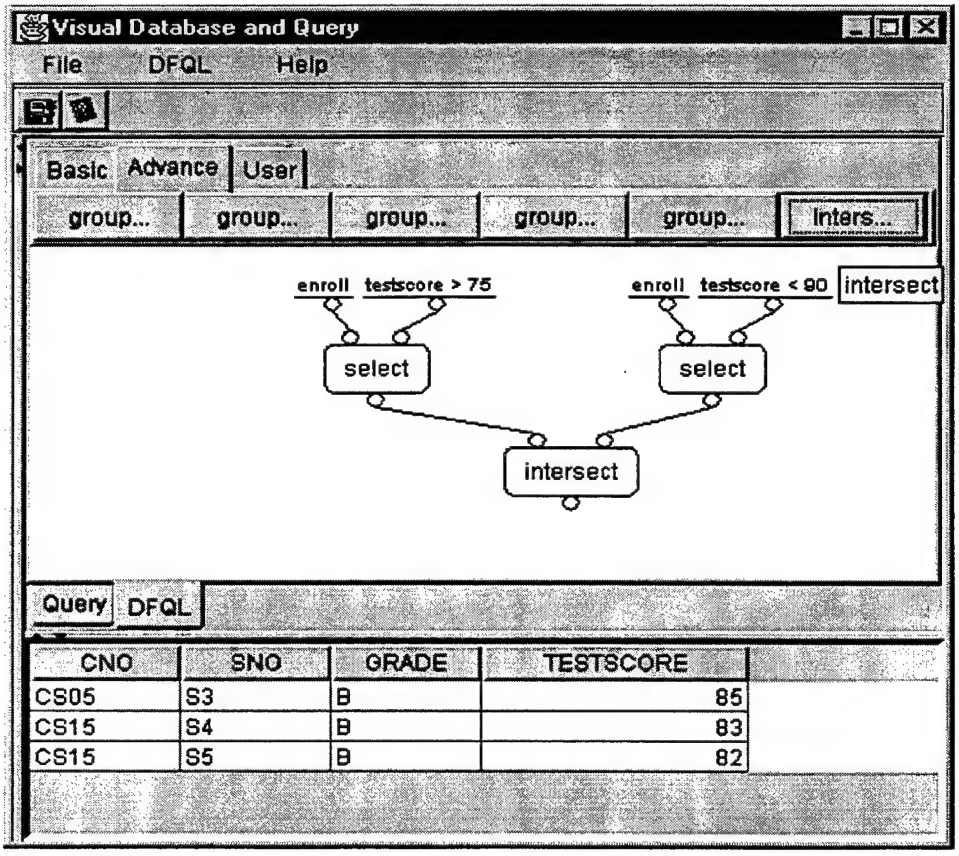


Figure 34. Execution of DFQL operator – Intersect

17. Execution of an incremental query

This example shows the hierarchical way to create an incremental query. First, the query choose all the records from "**course**" table with condition where "cno = 'CS10'. From above result, it joins with another table "instructor" with condition where both "ino" is the same. Finally, from the join result, display only the "iname" column information.

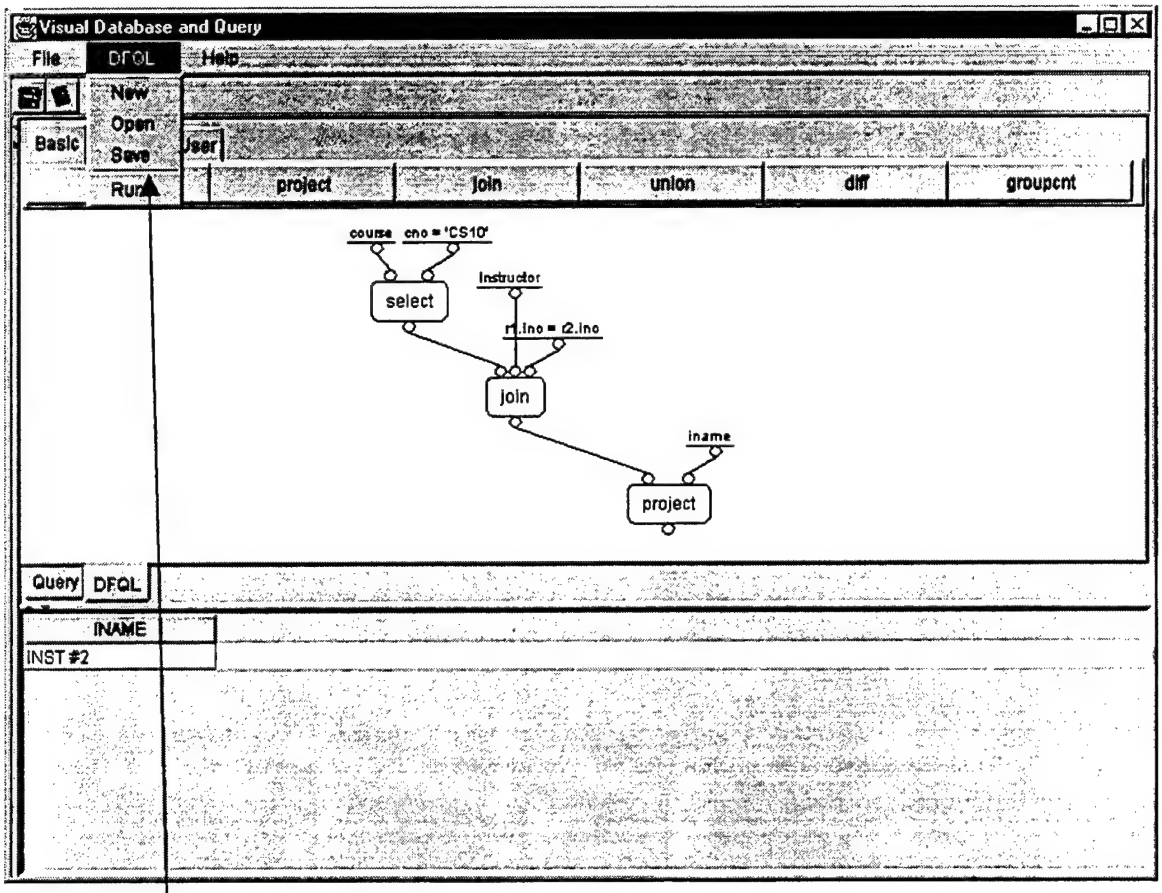


Figure 35. Execution of an incremental query

18. Execution of User Defined Operator

a) Create an user defined operator

This screen shows the initial step to create the user defined operator. By clicking the "**New**" button, the setup property window pops up. Enter the information as shown as shown. The user operator name is called "SelProj", with 1 relation, 1 condition, 1 attribute input.

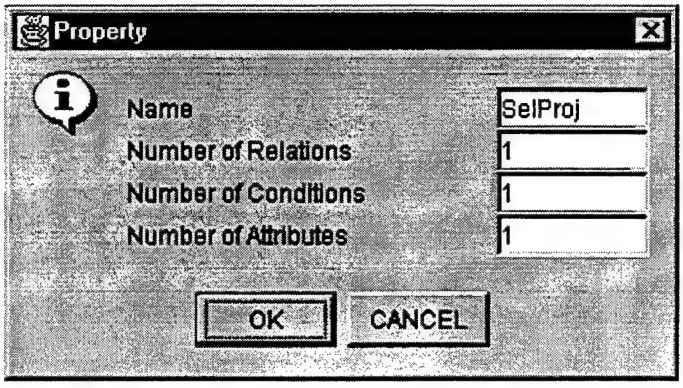


Figure 36. Initial Creation of New User Operator

b) Construct the operator

Accept the input by clicking "**OK**" button. An input bar shows up on the canvas. Go to "Basic" tab, select "**select**" and "**project**" operator. Arrange the operators on the comfortable area. Right click on "**project**" symbol, enter "**Operator4**" (or whatever the name of "select" symbol) on the "Relation" field in the property window.

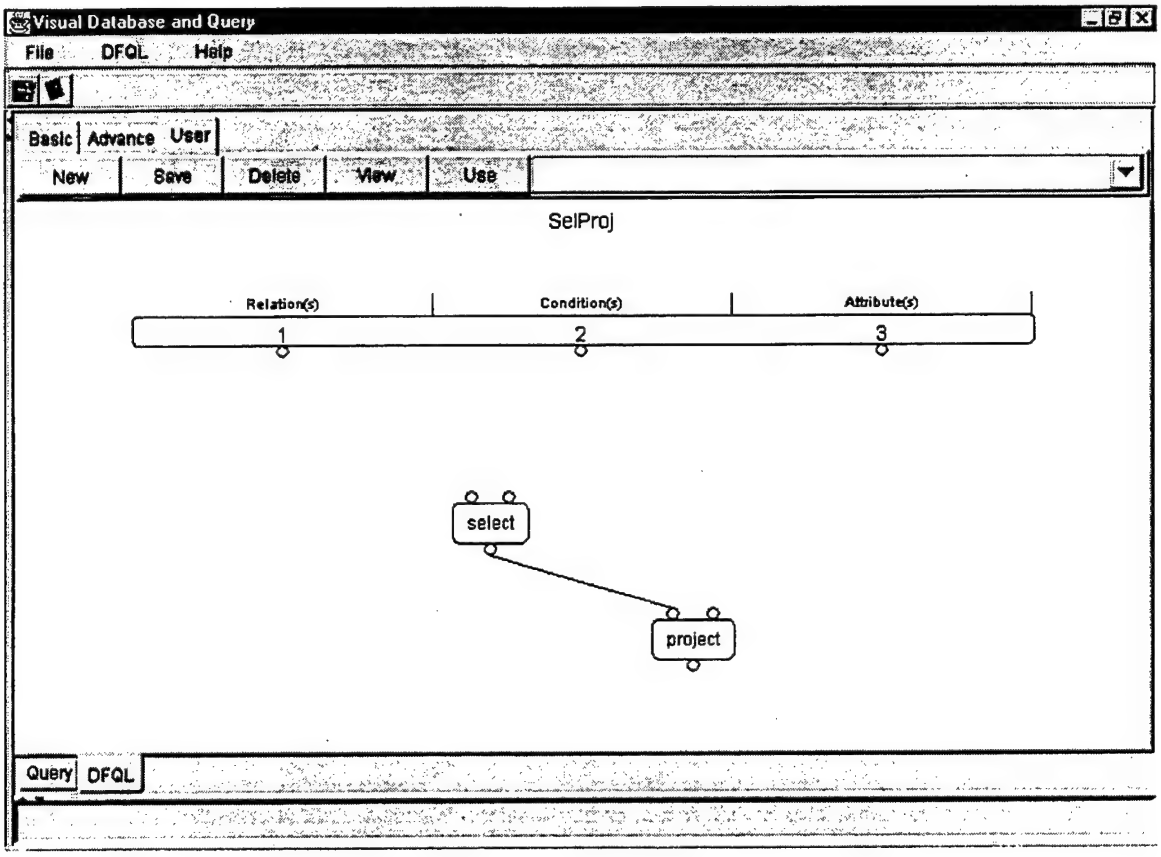


Figure 37. Operators linkage

Now, right click "1" on the input bar, a property window pops up. Enter "Operator4" (or whatever the name of the "**select**" symbol) on "Target Operator Name" field, "1" on "Target Operator Node" field. Do the same thing on node 2, 3 in the input bar. The screen shot shows the information on node 3.

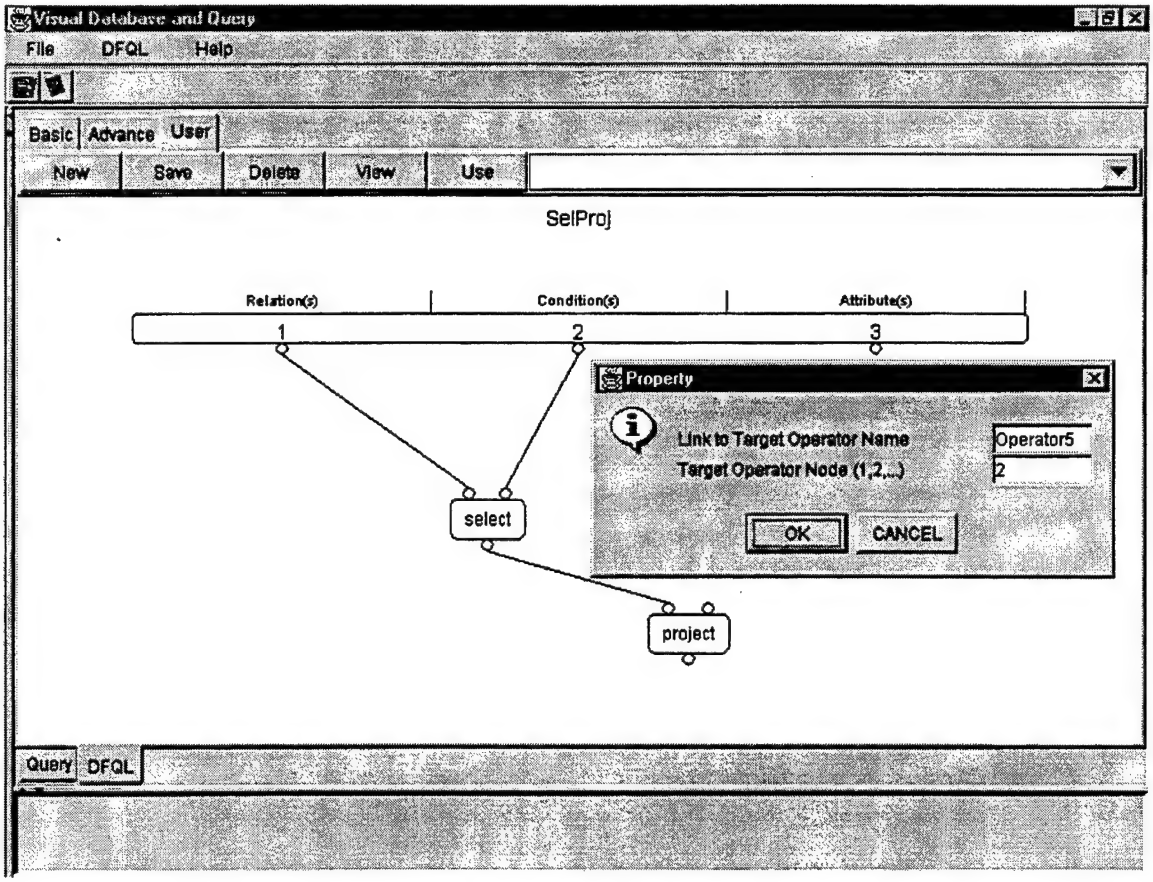


Figure 38. Input Node Setup

c) Save and View the user operator

After the all the input node information are setup, the information is ready to be saved. By clicking "**Save**" button, this newly constructed user operator is saved under the working directory in the hard disk. The file name is **SelProj.udo**. Also, SelProj shows up on ComboBox, means it is currently available for access.

To view the structure of the operator, select the name from the combobox, and click "**View**" button. The design diagram about this operator is shown on the screen.

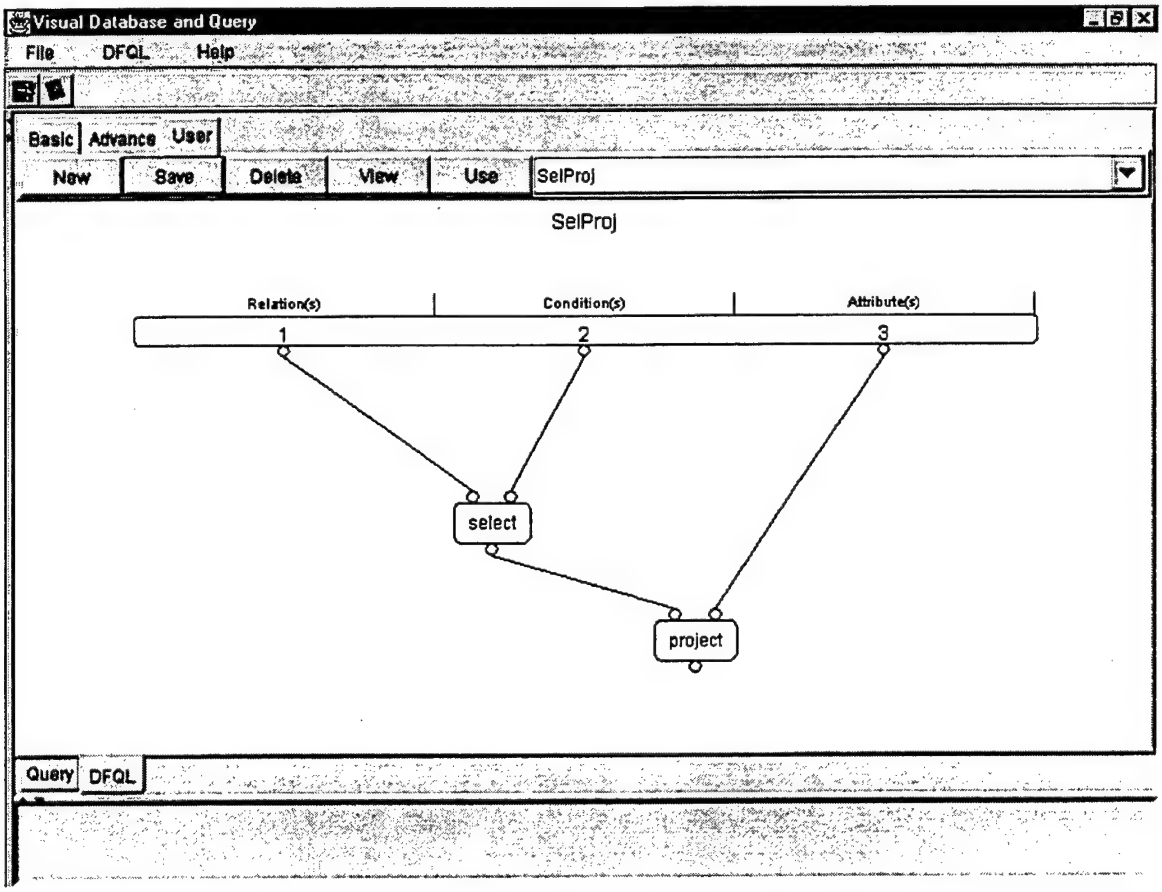


Figure 39. View the Existing User Defined Operator

d) Use and Run the User Defined Operator

Once the user defined operator is available, it just acts like the predefined operators. To use one, first select the name from the combobox, then click "Use" button. The symbol of this operator shows up on the canvas. Drag and drop the symbol to the comfortable area. Right click on the symbol area, a property window pops up. Input the information. Then from DFQL menu, select "Run". The result shows up on the table.

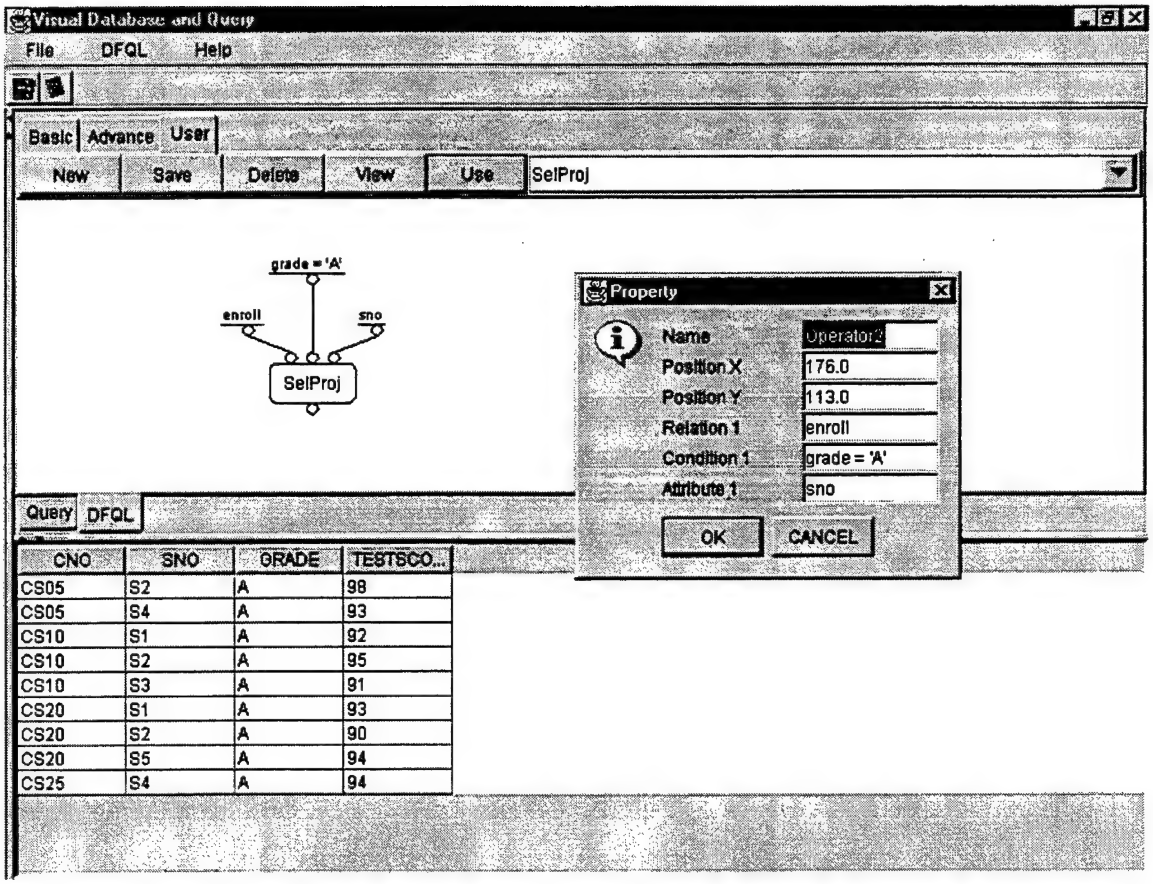


Figure 40. Use and Run the User Defined Operator

VI. CONCLUSION AND RECOMMENDATIONS

By far, all the basic functionalities on this application are fully tested and working properly base on the design requirement. The graphical user interface is quite simple, intuitive, and yet powerful and flexible. By providing the enough information (user name, password, database name, and driver), the user is able to log into any databases with proper setup. The user is able to view the database metadata through the tree view visually. The user is able to type in any regular queries on the text area, execute the query, view and sort the results on the table grid. Finally, the user is able to use the DFQL operators to build the complex query incrementally and graphically. The DFQL queries can be saved and opened at any time anywhere.

To make this application more flexible and powerful, several area of the improvement should be worth considering. Currently, DFQL canvas extends JPanel class to hold all the DFQL symbols in a query. This gives a limited space to hold the DFQL query, in other word, there are probably not enough room to hold a very complex query (e.g. 15 DFQL operators in the same canvas) on a 640x480 pixels screen. One of the solution is to extend JScrollPane class instead of JPanel class, but do requirement additional coding. In addition, each node of the operator can be extended to have its mouse listener, so that it can be dragged and dropped from one operator to another operator to establish the relationship. Currently, each operator has its own mouse listener, and mouse motion listener, and share the absolute position on the DFQL canvas. Each time one mouse movement is occurred, not only DFQL canvas receives this action,

but also all the operators receive the same action. This kind of situation makes the system work very busy. It is not very elegant way on object-oriented programming. To reduce and simplify the activities of the operator, the operator class can extend JComponent class, and implement the mouse listener, and mouse motion listener interface. In this way, each operator is a component of the DFQL canvas. When a mouse action occurs on the DFQL canvas, only the operator that is in the target area will receive the action. However, this requires tremendous re-coding on the operator class and its derived class.

APPENDIX. SOURCE CODE

Source code files are listed in file name alphabetical order.

1. AboutBox.java

```
/*
 * Author: Ron Chen
 * File: AboutBox.java
 * Last Modified: March 2, 1999
 *
 * A window class for About Window
 *
 *
 * Most of the code from this class are generated from
 * JBuilder 2.0 which no longer being used for the development
 * for this thesis project
 *
*/

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class AboutBox extends Dialog implements ActionListener{

  JPanel panel1 = new JPanel();
  JPanel panel2 = new JPanel();
  JPanel insetsPanel1 = new JPanel();
  JPanel insetsPanel2 = new JPanel();
  JPanel insetsPanel3 = new JPanel();
  JButton button1 = new JButton();
  JLabel label1 = new JLabel();
  JLabel label2 = new JLabel();
  JLabel label3 = new JLabel();
  JLabel label4 = new JLabel();
  BorderLayout borderLayout1 = new BorderLayout();
  BorderLayout borderLayout2 = new BorderLayout();
  FlowLayout flowLayout1 = new FlowLayout();
  FlowLayout flowLayout2 = new FlowLayout();
  GridLayout gridLayout1 = new GridLayout();
  String product = "Visual Database and Query";
  String version = "";
  String copyright = "Copyright (c) 1998";
  String comments = "Visual Database and Query";

  public AboutBox(Frame parent)
  {
    super(parent);
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try
    {
      jbInit();
    }
```

```
    catch (Exception e)
    {
      e.printStackTrace();
    }
    pack();
  }

  private void jbInit() throws Exception
  {
    this.setTitle("About");
    setResizable(false);
    panel1.setLayout(borderLayout1);
    panel2.setLayout(borderLayout2);
    insetsPanel1.setLayout(flowLayout1);
//    insetsPanel1.setBevelInner(JPanel.FLAT);
    insetsPanel2.setLayout(flowLayout1);
//    insetsPanel2.setMargins(new Insets(10, 10, 10, 10));
//    insetsPanel2.setBevelInner(JPanel.FLAT);
    gridLayout1.setRows(4);
    gridLayout1.setColumns(1);
    label1.setText(product);
    label2.setText("Author: Ron Chen");
    label3.setText(copyright);
    label4.setText("NPS MSCS - Theis Project");
    insetsPanel3.setLayout(gridLayout1);
//    insetsPanel3.setMargins(new Insets(10, 60, 10, 10));
//    insetsPanel3.setBevelInner(JPanel.FLAT);
    button1.setText("OK");
    button1.addActionListener(this);
//    insetsPanel2.add(imageControl1, null);
    panel2.add(insetsPanel2, BorderLayout.WEST);
    this.add(panel1, null);
    insetsPanel3.add(label1, null);
    insetsPanel3.add(label2, null);
    insetsPanel3.add(label3, null);
    insetsPanel3.add(label4, null);
    panel2.add(insetsPanel3, BorderLayout.CENTER);
    insetsPanel1.add(button1, null);
    panel1.add(insetsPanel1, BorderLayout.SOUTH);
    panel1.add(panel2, BorderLayout.NORTH);
  }

  protected void processWindowEvent(WindowEvent e)
  {
    if (e.getID() == WindowEvent.WINDOW_CLOSING)
    {
      cancel();
    }
    super.processWindowEvent(e);
  }

  void cancel()
  {
    dispose();
  }

  public void actionPerformed(ActionEvent e)
  {
```

```
    if (e.getSource() == button1)
    {
      cancel();
    }
  }
}
```

2. DB.java

```
/*
 * Author: Ron Chen
 * File: DB.java
 *
 * Wrapper Class for handling all the database activities
 *
*/

import java.util.Vector;
import java.util.*;
import java.sql.*;
import java.net.URL;
import javax.swing.*;
import javax.swing.text.*;
import javax.swing.tree.*;
import java.net.*;

public class DB
{
  private Connection con;
  // public Statement stmt;

  public boolean DBOK;

  // fill the table information
  public String[] columnNames = {};
  public Vector rows = new Vector();
  public ResultSet resultSet;
  public ResultSetMetaData metaData;

//  DefaultMutableTreeNode top;

  private String userName;
  private String password;
  private String server;
  private String driver;

  public DB()
  {
    // default user database is Microsoft Access Database

    userName = new String("Admin");
    password = new String("");
    server = new String("jdbc:odbc:Thesis");
    driver = new String("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
/* For Testing */
/*
    userName = new String("npscs");
    password = new String("npscs");
    server = new String("jdbc:odbc:csthesis");
    driver = new String("sun.jdbc.odbc.JdbcOdbcDriver");
*/
    try
    {
      DBOK = false;
      dbInit();
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }

  public DB(String PuserName, String Ppassword,
            String Pserver, String Pdriver) {
    userName = new String(PuserName);
    password = new String(Ppassword);
    server = new String(Pserver);
    driver = new String(Pdriver);

    try
    {
      DBOK = false;
      dbInit();
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }

  // Close the database connection
  public void closeConnection() throws Exception
  {
    try
    {
     if (!con.isClosed()) {
        System.out.println("Closing connection");
        con.close();
        // set to null
        con = null;
      }
    } catch (Exception e)
    {
      System.err.println("System Exception in closeConnection");
      System.err.println(e);
      throw e;
    }
  }  // end of closeConnection


  // Obtain the table name
  // This method is mainly used for testing metadata retrieving
```

```
  // the console screen
  public void PrintTableList()
  {
    boolean notdone = true;
    String types[] = {"TABLE", "VIEW", "SYSTEM TABLE", "GLOBAL",
                      "TEMPORARY", "LOCAL TEMPORARY", "ALIAS",
"SYNONYM"};

    String catalog = "";

    try
    {

      // get a DatabaseMetaData object
      System.out.println("getMetaData");
      DatabaseMetaData dmd = con.getMetaData();


      // Print Catalog Term
      System.out.println("Catalog term: " + dmd.getCatalogTerm());

      // Print Catalog
      catalog = con.getCatalog();
      System.out.println("Catalog name: " + catalog);

      String dbProductName = new String(dmd.getDatabaseProductName());
      if (dbProductName.indexOf("ACCESS") != -1) {
        catalog = getOdbcDsn();
        System.out.println("ODBC DSN = " + catalog);
      }


      // Print the schemas term
      System.out.println("schemas term: " + dmd.getSchemaTerm());

      // Print the schemas
/*      System.out.println("database schemas:");
      ResultSet rsSchemas = dmd.getSchemas();
      while (rsSchemas.next()) {
        System.out.println(rsSchemas.getString(1));
      }
      rsSchemas.close();
      rsSchemas = null;
*/
      // retrieve the User table info
      // System.out.println("Get all the table name");
      // ResultSet rs = dmd.getTables(con.getCatalog(),null,null,
types);
      System.out.println("Get all the table name belong to user " +
userName);
      ResultSet rs = null;
      if (dbProductName.indexOf("ACCESS") != -1) {
        rs = dmd.getTables(catalog,null,null, types);
      } else {
        rs = dmd.getTables(catalog,userName,null, types);
      }
//      ResultSet rs = dmd.getCatalogs();
      //ResultSet rs = dmd.getSchemas();
```

```
      System.out.println("finish getTables()");

      while (notdone)
      {
        notdone = rs.next();
        if (notdone)
        {
//          System.out.println("See if I can print the table name ");
          if (!(rs.getString(4).equals("SYSTEM TABLE")))
          {
            System.out.println(rs.getString(3));
          }
          // System.out.println(rs.getString(3));

//          String sLine = "Catelog: " + rs.getString(1);
//          sLine = sLine + " Schema: " + rs.getString(2);
//          sLine = sLine + " Table_Name " + rs.getString(3);
//          sLine = sLine + " Table_Type " + rs.getString(4);
//          sLine = sLine + " Remarks = " + rs.getString(5);

//          System.out.println(sLine);
        }  // end of it
      } // end of while

      // Close the ResultSet
      rs.close();

    } catch(Exception e)
    {
      System.out.println(e);
    }
  }  // end of PrintTableList


  public DefaultMutableTreeNode fillDbMetaData() {
    DefaultMutableTreeNode top = null;

    // List all the table name
    boolean notdone = true;
    //String types[] = {"TABLE", "VIEW", "SYSTEM TABLE", "GLOBAL",
    //                  "TEMPORARY", "LOCAL TEMPORARY", "ALIAS",
"SYNONYM"};
    //Only show User tables and Views
    String types[] = {"TABLE", "VIEW"};

   System.out.println("fillDbMetaData()");

    try
    {
      // Get the catelog name
      String catalog = new String(con.getCatalog());


      // get a DatabaseMetaData object
      System.out.println("getMetaData");
      if (con.isClosed())
        System.out.println("Connection is closed");
      else
```

```
            System.out.println("Connection is not closed");

        DatabaseMetaData dmd = con.getMetaData();

        String dbProductName = new String(dmd.getDatabaseProductName());
        System.out.println("Product Name: " + dbProductName);

        // retrieve the User table info
        ResultSet rs = null;
        if (dbProductName.indexOf("ACCESS") != -1) {
          // Obtain the ODBC DSN for this database
          catalog = getOdbcDsn();
          // special case for ACCESS database
          rs = dmd.getTables(catalog,null,null, types);
        } else {
          rs = dmd.getTables(catalog,userName,null, types);
        }

        System.out.println("Create the root node");
        TreeNodeName dbName = new TreeNodeName("Database", "Database -
root");
        top = new DefaultMutableTreeNode(dbName);

        // Table
        TreeNodeName tblRoot = new TreeNodeName("Table", "Table List");
        DefaultMutableTreeNode tbl = new DefaultMutableTreeNode(tblRoot);

        // View
        TreeNodeName vwRoot = new TreeNodeName("View", "View List");
        DefaultMutableTreeNode vw = new DefaultMutableTreeNode(vwRoot);

        while (notdone)
        {
          notdone = rs.next();
          if (notdone)
          {
            //System.out.println("See if I can print the table name ");
            // Get Table Type
            String tableType = new String(rs.getString(4));
            if (tableType.equals("TABLE"))
            {
              // Get the table name
              String tblnm = new String(rs.getString(3));

              // Tool Tips String
              StringBuffer toolTips = new StringBuffer();

              System.out.println("Get the primary keys");
              // Get unique columns - Primary Key
              ResultSet rsPK = null;
              if (dbProductName.indexOf("ACCESS") != -1) {
                // special case for access database
                // System.out.println("Access MDB dsn = " + catalog);
                // Access ODBC driver does not support this function
                // rsPK = dmd.getPrimaryKeys(catalog,null,tblnm);
              } else {
                rsPK = dmd.getPrimaryKeys(null,userName,tblnm);
              }
```

```
            boolean bFirst = true;

            // System.out.println("Loop the primary key resultset");
            if (rsPK != null) {
              while (rsPK.next()) {
                // Get the primary key name
                if (bFirst) {
                  toolTips.append("Primary Key Name: " );
                  toolTips.append(new String(rsPK.getString(6)));
                  toolTips.append("\n");
                  // Don't get the second time
                  bFirst = false;
                }

                // Get the Column Name
                toolTips.append("Column Name: ");
                toolTips.append(new String(rsPK.getString(4)));
                // Get the Key Sequence
                toolTips.append("  (Key Sequence: ");
                toolTips.append(rsPK.getShort(5));
                toolTips.append(")\n");
              }

              rsPK.close();
              rsPK = null;
            }

            /* Getting indexing information takes too long, no good.

            // Get the index information
            ResultSet rsIndex = null;
            if (dbProductName.indexOf("ACCESS") != -1) {
              // special case for access database
              // Access 97 ODBC driver does not support this function
              // rsIndex = dmd.getIndexInfo(catalog, null, tblnm, false,
false);
            } else {
              rsIndex = dmd.getIndexInfo(null, userName, tblnm, false,
false);
            }

            StringBuffer indexInfo = new StringBuffer("");

            bFirst = true;
            String holdIndexName = "";
            if (rsIndex != null) {
              while (rsIndex.next()) {
                if (bFirst) {
                  System.out.println("Get Index Information");
                  indexInfo.append("\nIndex Column(s):\n");
                  bFirst = false;
                }

                // Get Index type
                short type = rsIndex.getShort(7);
                if (type != DatabaseMetaData.tableIndexStatistic) {
                  String currentIndexName = new
String(rsIndex.getString(6));
```

```
                    if ((holdIndexName.length() == 0) ||
(!(holdIndexName.equalsIgnoreCase(currentIndexName)))) {
                      // hold the current value
                      holdIndexName = currentIndexName;

                      indexInfo.append("Index Name: ");
                      indexInfo.append(currentIndexName);
                      indexInfo.append("\n");
                    }


                    // Column name
                    indexInfo.append(new String(rsIndex.getString(9)));
                    indexInfo.append("   Order: ");
                    // Sort by (Ascending or Descending)
                    indexInfo.append(new String(rsIndex.getString(10)));
                    // Ordinal poistion
                    indexInfo.append("   Ordinal Position: ");
                    indexInfo.append(rsIndex.getString(8)).append("\n");
                  }
                }
                // Close the resultset
                rsIndex.close();
                rsIndex = null;
              }

              // Put the index information to tooltip string buffer
              if (indexInfo.length() > 0) {
                toolTips.append(new String(indexInfo.toString()));
              }

              */


              // Builder the Tree Node with the Tool Tip Text
              TreeNodeName NodeName = new
TreeNodeName(tblnm,toolTips.toString());

              // Create the Tree Node
              // DefaultMutableTreeNode treeTblNm = new
DefaultMutableTreeNode(tblnm);
              DefaultMutableTreeNode treeTblNm = new
DefaultMutableTreeNode(NodeName);
              tbl.add(treeTblNm);


              // Following lines fixes the multiple words table/query name
              int iPos = tblnm.indexOf(' ');
              if (iPos >= 0)
              {
                System.out.println(tblnm + " contains empty char");
                String newnm = new String("[" + tblnm + "]");
                tblnm = new String(newnm);
                System.out.println("new table name = " + tblnm);
              }
              System.out.println("Table name = " + tblnm);
```

```
            ResultSet rsColumn = null;
            if (dbProductName.indexOf("ACCESS") != -1) {
              // this is Access MDB
              rsColumn = dmd.getColumns(catalog, null, tblnm, null);
            } else {
              rsColumn = dmd.getColumns(null, null, tblnm, null);
            }

            boolean bNotDoneLooping = false;
            if (rsColumn != null) {
              bNotDoneLooping = rsColumn.next();
            }

            if (bNotDoneLooping == false)
              System.out.println(tblnm + " has no columns");

            while (bNotDoneLooping)
            {
              // Obtain the Column name
              String columnName = new String(rsColumn.getString(4));
              String columnTypeName = new String(rsColumn.getString(6));
              int columnSize = rsColumn.getInt(7);

              // Create the Node with ToolTip
              TreeNodeName colNodeName = new TreeNodeName(columnName,
                                                  columnTypeName + "(" +
columnSize + ")");

              DefaultMutableTreeNode treeColNm = new
DefaultMutableTreeNode(colNodeName);
              treeTblNm.add(treeColNm);
              // Move to next column
              bNotDoneLooping = rsColumn.next();
            }
            rsColumn.close();
            rsColumn = null;

            // System.out.println(rs.getString(3));
          }

          if (tableType.equals("VIEW"))
          {
            DefaultMutableTreeNode vwNm = new
DefaultMutableTreeNode(rs.getString(3));
            vw.add(vwNm);

            // System.out.println(rs.getString(3));
          }


        }  // end of it
      } // end of while

      // Add two subtree
      top.add(tbl);
      top.add(vw);

      // Close the resultset
```

```
      rs.close();
      rs = null;

      dmd = null;

    } catch(Exception e)
    {

      System.out.println(e);
    }

    /*
    try {
      System.out.println("test: create a statement");
      Statement st = con.createStatement();
      System.out.println("test: execute query");
      ResultSet rsTest = st.executeQuery("select * from States");
      if (rsTest.next()) {
        System.out.println("first record, first column: " +
rsTest.getString(1));
      }
      rsTest.close();
    } catch(SQLException se)
    {

      System.out.println(se);
    }
    */


    System.out.println("return the node");
    return top;
  }


  // Retrieve Drive information
  public void viewDrivers()
  {
    System.out.println("Sarting enum");
    Enumeration e =  DriverManager.getDrivers();
    while (e.hasMoreElements())
    {
      java.sql.Driver d = (java.sql.Driver) e.nextElement();
      System.out.println("Driver name: " + d.getClass().getName());
    }
  }

  // Obtain the Data source name for
  public String getOdbcDsn() {
    int n = server.indexOf("jdbc:odbc:");
    if (n != -1) {
      // found, this connection is using JDBCODBC driver
      StringBuffer dsn = new StringBuffer(server);
      return new String(dsn.substring(10));
    } else {
      return new String("");
    }
  }
```

```
  // Execute the query
  public boolean executeQuery(String query) {
    Statement stmt;

    if (con == null) {
      System.err.println("There is no database to execute the query.");
      return false;
    }

    try {
      System.out.println("Test if the connection is closed");
      if (con.isClosed()) {
        System.err.println("Database connection is closed");
        return false;
      }
      // Create a statement of the database
      System.out.println("create a statement");
      stmt = con.createStatement();

      System.out.println("excute the query: " + query);
      resultSet = stmt.executeQuery(query);
      /*
      stmt.execute(query);
      System.out.println("Get the ResultSet");
      resultSet = stmt.getResultSet();
      */
      System.out.println("Get the Meta Data");
      metaData = resultSet.getMetaData();

      int numberOfColumns =  metaData.getColumnCount();
      columnNames = new String[numberOfColumns];
      // Get the column names and cache them.
      // Then we can close the connection.
      for(int column = 0; column < numberOfColumns; column++) {
        columnNames[column] = metaData.getColumnLabel(column+1);
      }

      // Get all rows.
      rows = new Vector();
      while (resultSet.next()) {
        Vector newRow = new Vector();
        for (int i = 1; i <= numberOfColumns; i++) {
          // Create a new object value, instead of reference it
          newRow.addElement(resultSet.getObject(i));
          //newRow.addElement(new
String(resultSet.getObject(i).toString()));
        }
        rows.addElement(newRow);
      }
      return true;
    }
    catch (SQLException ex) {
      System.err.println(ex);
      return false;
    }
  }
```

```
  public void dbInit() throws Exception
  {
    System.out.println("Set the connection of the database");

     // Assume the database can't be open
    DBOK = false;

    // Create connection
    System.out.println("Obtain the connection object");
    // This sun default jdbc-odbc bridge driver not good enough
    // to recognize the metadata
    // Use more sophisticated one
    Class.forName(driver);

    try {
       con = DriverManager.getConnection(server, userName, password);
    } catch(SQLException e) {
      System.err.println("DB->dbInit(), error: ");
      System.err.println(e);
      return;
    }

    // Some debugging lines
    System.out.println("Check SQL supporting level:");
    System.out.println("Entry Level? "
                       +
con.getMetaData().supportsANSI92EntryLevelSQL());
    System.out.println("Intermediate Level? "
                       +
con.getMetaData().supportsANSI92IntermediateSQL());
    System.out.println("Full Level? "
                       + con.getMetaData().supportsANSI92FullSQL());


    // Database is successfully open
    DBOK = true;
  }


  // Testing routine
  public static void main(String argv[])
  {
    DB odb;
    try
    {
        odb = new DB();
        odb.viewDrivers();
        odb.PrintTableList();

        odb.closeConnection();
    } catch (Exception e) {
        System.err.println(e);
    }
  }

}
```

3. DFQL.java

```
/*
 * Author: Ron Chen
 * File: DFQL.java
 *
 * A window class for handling DFQL panel
 *
*/

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.filechooser.*;
import java.io.*;
import java.util.*;


public class DFQL extends JPanel {

  // constants for user operators
  // INUSED means that the DFQL canvas is doing regular DFQL
construction
  // DESIGN means that the DFQL canvas is doing the user defined
operator construction
  public final static int INUSED = 1;
  public final static int DESIGN = 2;

  // Filter files for File Chooser Dialog
  ExampleFileFilter dfqFilter = new ExampleFileFilter("dfq", "Data Flow
Query");
  ExampleFileFilter allFilter = new ExampleFileFilter("*", "All Files");

  String[] basicOperatorLabels = {"select", "project", "join",
                                  "union", "diff", "groupcnt"};

  String[] basicOperatorsToolTips = {"SELECT DISTINCT * FROM relation
WHERE condition",
                                     "SELECT DISTINCT attribute list
FROM relation",
                                     "SELECT DISTINCT * FROM relation
r1, relation r2 " +
                                     "WHERE join condition",
                                     "SELECT DISTINCT * FROM relation1
UNION " +
                                     "SELECT DISTINCT * FROM relation2",
                                     "SELECT DISTINCT * FROM relation1
MINUS " +
                                     "SELECT DISTINCT * FROM relation2",
                                     "SELECT DISTINCT grouping
attributes COUNT(*) " +
                                     "count attributes FROM relation " +
                                     "GROUP BY grouping attributes"};

  JButton[] basicOperators = new JButton[basicOperatorLabels.length];

  String[] advanceOperatorLabels = {"groupAllsatisfy", "groupNsatisfy",
```

```
                                        "groupmax", "groupmin", "groupavg",
"intersect"};

  String[] advanceOperatorToolTips = {"group all satisfy",
                                      "group all satisfy for N records",
                                      "group maximum", "group mininum",
                                      "group average", "intersect"};

  JButton[] advanceOperators = new
JButton[advanceOperatorLabels.length];


  String[] userDefinedLabels = {"New", "Save", "Delete", "View", "Use"};
  String[] userDefinedToolTips = {"Create a new operator",
                                  "Save the current operator",
                                  "Delete the current operator",
                                  "View the current operator design",
                                  "Use the current user defined
operator"};

  JButton[] userDefinedAction = new JButton[userDefinedLabels.length];

  JTabbedPane tabbedPaneDFQL;

  // Panel on the first tab of the tabbedPaneDFQL
  JPanel panelBasicOperator;

  // Panel on the second tab of the tabbedPaneDFQL
  JPanel panelAdvanceOperator;

  // Panel on the third tab of the tabbedPaneDFQL
  JPanel panelUserOperator;

  // Panel that holds all the user defined action buttons
  JPanel panelUserDefinedAction;
  // Combobox holds all the user defined operator names
  JComboBox comboUserOperators;


  // Panel on the Center for holding DFQL operation
  DFQLCanvas panelOperation;


  // Collection of DFQL operators
  Vector vDFQLOperators = new Vector();

  // Collection of operators in the User defined Operator
  Vector vUserOperators = new Vector();

  // list of User Operators
  Vector vUserOperatorsList = new Vector();

  // Current User Operator
  OperatorUser opUser = null;

  // mode on the current DFQL process
  int mode = INUSED;              // default as INUSED.
```

```
  public DFQL() {


    /* Basic Panel */
    panelBasicOperator = new JPanel(new
GridLayout(0,basicOperatorLabels.length));
    for (int i=0; i<basicOperatorLabels.length; ++i) {
      basicOperators[i] = new JButton(basicOperatorLabels[i]);
      basicOperators[i].setToolTipText(basicOperatorsToolTips[i]);
      basicOperators[i].addActionListener(new ListenerBasicOperators());
      panelBasicOperator.add(basicOperators[i]);
    }



    /* Advance Panel */
    panelAdvanceOperator = new JPanel(new
GridLayout(0,advanceOperatorLabels.length));
    for (int i=0; i<advanceOperatorLabels.length; ++i) {
      advanceOperators[i] = new JButton(advanceOperatorLabels[i]);
      advanceOperators[i].setToolTipText(advanceOperatorToolTips[i]);
      advanceOperators[i].addActionListener(new
ListenerAdvanceOperators());
      panelAdvanceOperator.add(advanceOperators[i]);
    }

    /* User Defined Panel */
    panelUserOperator = new JPanel(new BorderLayout());
    panelUserDefinedAction = new JPanel(new GridLayout(0,
userDefinedLabels.length));
    for (int i=0; i<userDefinedLabels.length; i++) {
       userDefinedAction[i] = new JButton(userDefinedLabels[i]);
       userDefinedAction[i].setToolTipText(userDefinedToolTips[i]);
       userDefinedAction[i].addActionListener(new
ListenerUserDefinedAction());
       panelUserDefinedAction.add(userDefinedAction[i]);
    }
    comboUserOperators = new JComboBox();
    // For testing only
    //comboUserOperators.addItem("SelProj");
    // List all the user operator in the working direcotry to the
ComboBox
    listUserOperators();
    panelUserOperator.add(panelUserDefinedAction, BorderLayout.WEST);
    panelUserOperator.add(comboUserOperators, BorderLayout.CENTER);



    /* Operation Panel */
    panelOperation = new DFQLCanvas();
    // Make the reference to collection object
    panelOperation.refDFQLOperators = vDFQLOperators;

    /* Tabs layout */
    tabbedPaneDFQL = new JTabbedPane();
    tabbedPaneDFQL.addTab("Basic", panelBasicOperator);
```

```
    tabbedPaneDFQL.addTab("Advance", panelAdvanceOperator);
    tabbedPaneDFQL.addTab("User", panelUserOperator);
    tabbedPaneDFQL.setTabPlacement(JTabbedPane.TOP);
    tabbedPaneDFQL.setMinimumSize(new Dimension(10,10));

    // set the layout
    this.setLayout(new BorderLayout());
    this.add(panelOperation, BorderLayout.CENTER);
    this.add(tabbedPaneDFQL, BorderLayout.NORTH);
  }


  // getQuery() - Construct a query base on DFQL diagram
  public String getQuery() {
    // Check if there is any objects in the collection object
    if (vDFQLOperators.isEmpty()) {
       return (new String(""));
    }

    // Get the last operator in the collection
    Operator op = (Operator) vDFQLOperators.lastElement();
    // Construct the Query from back
    return getQuery(op);
  }

  // With operator object as parameter
  public String getQuery(Operator op) {
    if (op == null) {
      return (new String(""));
    }

    return (new String(op.buildQuery()));
  }


  // New DFQL section
  public void newDFQL() {
    // reset the mode
    mode = INUSED;

    // remove all the objects in the collection object
    vDFQLOperators.clear();
    // clear the operation panel
    panelOperation.clear();
  }

  // Open Exist DFQL from disk
  public void openDFQL() {
    String inFile = "";

    JFileChooser chooser = new JFileChooser(getWorkingDirectory());
    chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
    // add the filter
    chooser.addChoosableFileFilter(dfqFilter);
    // make this filter as the current file filter
    chooser.setFileFilter(dfqFilter);
      int retval = chooser.showOpenDialog(this);
```

```
      if(retval == JFileChooser.APPROVE_OPTION) {
          File theFile = chooser.getSelectedFile();
          if(theFile != null) {
                if(theFile.isFile()) {
                  inFile = chooser.getSelectedFile().getAbsolutePath();
          }
              }
      }

    // If no file is selected, then exit
    if (inFile.length() == 0) {
      return;
    }

    // Save the objects in the collection
    try {
      ObjectInputStream in = new ObjectInputStream(
                                    new FileInputStream(inFile));
      vDFQLOperators = (Vector) in.readObject();
      in.close();
    } catch(Exception e) {
      e.printStackTrace();
    }

    // This is important:
    // reset the vDFQLOperators reference in each operator object
    for (Enumeration e=vDFQLOperators.elements(); e.hasMoreElements();)
{

      Operator op = (Operator) e.nextElement();
      // reset the vector object reference
      op.refDFQLOperators = vDFQLOperators;
      // re-reimplement the all the interface
      // Add the events for the object
      panelOperation.addMouseListener(op);
      panelOperation.addMouseMotionListener(op);

      // Check if this operator is OperatorUser class
      if (op.getClass().getName().equalsIgnoreCase("OperatorUser")) {
        //System.out.println("DFQL->openDFQL(), This is OperatorUser
Object");
        vUserOperators = ((OperatorUser) op).vRefUserOperator;
        //System.out.println("DFQL->openDFQL(), finish setting
reference");
      }
    }

    // Check the User Operator collection,
    // make each operator inside this collection reference to this
collection
    for (Enumeration e=vUserOperators.elements(); e.hasMoreElements();)
{

      Object ob = e.nextElement();
      if (ob.getClass().getName().indexOf("Operator") != -1) {
      // reset the vector object reference
        Operator op = (Operator) ob;
        op.refDFQLOperators = vUserOperators;
```

```
        }
      }


      // System.out.print("Draw the objects");

      // reset the reference to vector object
      panelOperation.refDFQLOperators = vDFQLOperators;

      // repaint the objects in the collection
      panelOperation.repaint();

      // Add the Key listener to its component
      addKeyListener(panelOperation);
    }

    // Save the DFQL to disk
    public void saveDFQL() {
      String outFile = "";

      JFileChooser chooser = new JFileChooser(getWorkingDirectory());
      chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
      // add the filter
      chooser.addChoosableFileFilter(dfqFilter);
      // make this filter as the current file filter
      chooser.setFileFilter(dfqFilter);
        int retval = chooser.showSaveDialog(this);
        if(retval == JFileChooser.APPROVE_OPTION) {
            outFile = new
String(chooser.getSelectedFile().getAbsolutePath());
        }

      // If no file is selected, then exit
      if (outFile.length() == 0) {
        return;
      }

      System.out.println("Save to file = " + outFile);

      // Save the objects in the collection
      try {
        ObjectOutputStream out = new ObjectOutputStream(
                                      new FileOutputStream(outFile));
        out.writeObject(vDFQLOperators);
        out.close();
      } catch(Exception e) {
        e.printStackTrace();
      }
    }


    // Obtain the directory where the program is running
    public String getWorkingDirectory() {
      String wd = System.getProperty("user.dir");

      if (wd != null) {
        return (new String(wd));
```

```
    }

    return (new String(""));
  }

  // List all the User operators into the comboBox
  private void listUserOperators() {
    // Obtain the list of all the user defined operator files
    // under the working directory
    try {
      File path = new File(getWorkingDirectory());
      // System.out.println("current path = " + path.getPath());
      String[] list = path.list(new FileType("udo"));

      if (list.length == 0) {
        // System.out.println("User operator list is empty");
        return;
      }
      // System.out.println("length of list = " + list.length);

      for (int i=0; i<list.length; i++) {
        System.out.println("User Operator file = " + list[i]);
        String sb = new String(list[i]);
        int pos = sb.indexOf(".udo");
        String operatorName = null;
        if (pos != -1) {
          operatorName = sb.substring(0, pos);
        }
        comboUserOperators.addItem(operatorName);
      }
     } catch (Exception e) {
        e.printStackTrace();
     }

  }

  // ------------------- Inner Class section -----------------

  // inner class for file filter base on the file type
  public class FileType implements FilenameFilter {
    String afn;
    FileType(String ft) {afn = ft;}
    public boolean accept(File dir, String name) {
      // Strip path information
      String f = new File(name).getName();
      return f.indexOf(afn) != -1;
    }
  }

  // inner class for basic operators listener
  public class ListenerBasicOperators implements ActionListener {

    public void actionPerformed(ActionEvent e) {
      // Point to different vector base on the mode value
      Vector vRefCollection = null;
      if (mode == INUSED) {
        vRefCollection =  vDFQLOperators;
      } else {
```

```
        vRefCollection = vUserOperators;
      }


      Operator op = new Operator();



      Object source = e.getSource();
      if (source == basicOperators[0])          // select
      {
        OperatorSelect opSelect = new OperatorSelect(vRefCollection);
        op = (Operator) opSelect;

      }
      else if (source == basicOperators[1])     // project
      {
        OperatorProject opProject = new OperatorProject(vRefCollection);
        op = (Operator) opProject;
      }
      else if (source == basicOperators[2])     // join
      {
        OperatorJoin opJoin = new OperatorJoin(vRefCollection);
        op = (Operator) opJoin;
      }
      else if (source == basicOperators[3])     // union
      {
        OperatorUnion opUnion = new OperatorUnion(vRefCollection);
        op = (Operator) opUnion;
      }
      else if (source == basicOperators[4])     // diff
      {
        OperatorDiff opDiff = new OperatorDiff(vRefCollection);
        op = (Operator) opDiff;
      }
      else if (source == basicOperators[5])     // groupcnt
      {
        OperatorGroupcnt opGroupcnt = new
OperatorGroupcnt(vRefCollection);
        op = (Operator) opGroupcnt;

      }


      // Draw the object
      op.draw(panelOperation.getGraphics());
      // Add the events for the object
      panelOperation.addMouseListener(op);
      panelOperation.addMouseMotionListener(op);
      // Reference to the proper collection object
      panelOperation.refDFQLOperators = vRefCollection;

      // Place this object into the collection object
      vRefCollection.add(op);

    }
  }
```

```
  // inner class for advance operators listener
  public class ListenerAdvanceOperators implements ActionListener {

    public void actionPerformed(ActionEvent e) {
      // Point to different vector base on the mode value
      Vector vRefCollection = null;
      if (mode == INUSED) {
        vRefCollection =  vDFQLOperators;
      } else {
        vRefCollection = vUserOperators;
      }

      Operator op = new Operator();

      Object source = e.getSource();
      if (source == advanceOperators[0])     // groupALLsatisfy
      {
        OperatorGroupALLsatisfy opGroupALLsatisfy = new
OperatorGroupALLsatisfy(vRefCollection);
        op = (Operator) opGroupALLsatisfy;
      }
      else if (source == advanceOperators[1])     // groupNsatisfy
      {
        OperatorGroupNsatisfy opGroupNsatisfy = new
OperatorGroupNsatisfy(vRefCollection);
        op = (Operator) opGroupNsatisfy;
      }
      else if (source == advanceOperators[2])     // groupmax
      {
        OperatorGroupmax opGroupmax = new
OperatorGroupmax(vRefCollection);
        op = (Operator) opGroupmax;
      }
      else if (source == advanceOperators[3])     // groupmin
      {
        OperatorGroupmin opGroupmin = new
OperatorGroupmin(vRefCollection);
        op = (Operator) opGroupmin;
      }
      else if (source == advanceOperators[4])     // groupavg
      {
        OperatorGroupavg opGroupavg = new
OperatorGroupavg(vRefCollection);
        op = (Operator) opGroupavg;

      }
       else if (source == advanceOperators[5])     // intersect
      {
        OperatorIntersect opIntersect = new
OperatorIntersect(vRefCollection);
        op = (Operator) opIntersect;

      }
```

```
      // Draw the object
      op.draw(panelOperation.getGraphics());
      // Add the events for the object
      panelOperation.addMouseListener(op);
      panelOperation.addMouseMotionListener(op);
      // Reference to the proper collection object
      panelOperation.refDFQLOperators = vRefCollection;

      // Place this object into the collection object
      vRefCollection.add(op);

    }
  }

  // inner class for user defined operators listener
  public class ListenerUserDefinedAction implements ActionListener {
    public void actionPerformed(ActionEvent e) {


      Object source = e.getSource();
      if (source == userDefinedAction[0])          // New
      {
        // Clean all the object inside the collection first
        vUserOperators.clear();
        // Clear the canvas
        panelOperation.clear();

        // User wants to design a new operator
        mode = DFQL.DESIGN;

        // Create the instance of the User Operator object
        opUser = new OperatorUser();
        // Set the mode of the user operator
        opUser.setMode(mode);
        // Set the drawing canvas reference
        opUser.setDesignCanvas(panelOperation);

        // Set the collection object reference
        opUser.vRefUserOperator = vUserOperators;
        panelOperation.refDFQLOperators = vUserOperators;

        // Set the object reference
        panelOperation.refOpUser = opUser;

        // Build the new operator
        opUser.newOperator();

         // Add the events for the object
        panelOperation.addMouseListener(opUser);
        panelOperation.addMouseMotionListener(opUser);



      }
      else if (source == userDefinedAction[1])     // Save
      {
        // Save the User Defined Operator (udo)
        String outFile = opUser.designName + ".udo";
```

```
        // Save the User objects
        try {
          ObjectOutputStream out = new ObjectOutputStream(
                                        new FileOutputStream(outFile));
          out.writeObject(opUser);
          out.close();
        } catch(Exception ex) {
          ex.printStackTrace();
        }
        // Add this name to the ComboBox if it is not existing
        comboUserOperators.removeItem(opUser.designName);
        comboUserOperators.addItem(opUser.designName);

      }
      else if (source == userDefinedAction[2])     // Delete
      {
        // Delete the selected user operator
        String operatorName = (String)
comboUserOperators.getSelectedItem();
        comboUserOperators.removeItem(operatorName);
        // Delete the file on the hard drive
        File file = new File(operatorName+".udo");
        file.delete();

      }
      else if (source == userDefinedAction[3])     // View
      {
        // Clean all the object inside the collection first
        vUserOperators.clear();
        // Clear the canvas
        panelOperation.clear();


        // Retrieve stored information from the file
        String operatorName = (String)
comboUserOperators.getSelectedItem();
        // extension "udo" means User Defined Operator
        String inFile = operatorName + ".udo";

        // Save the objects in the collection
        try {
          ObjectInputStream in = new ObjectInputStream(
                                        new FileInputStream(inFile));
          opUser = (OperatorUser) in.readObject();
          in.close();
        } catch(Exception ex) {
          ex.printStackTrace();
        }

        // Set the extra information before it really can be used
        // this is in DESIGN mode
        mode = DFQL.DESIGN;

        // Set the mode of the user operator
        opUser.setMode(mode);
        // Set the drawing canvas reference
        opUser.setDesignCanvas(panelOperation);
```

```
        // Set the collection object reference
        // Current user operator vector reference that one from the file
        vUserOperators = opUser.vRefUserOperator;
        // Reference to the same vector
        panelOperation.refDFQLOperators = vUserOperators;

        // Set the object reference
        panelOperation.refOpUser = opUser;

        // loop through each operator, and reset the some run time
properties
        Object ob = null;

        for (Enumeration en=vUserOperators.elements();
en.hasMoreElements();) {
          ob = en.nextElement();
          if (ob.getClass().getName().indexOf("Operator") != -1) {
            Operator op = (Operator) ob;
            op.refDFQLOperators = vUserOperators;
            // re-reimplement the all the interface
            // Add the events for the object
            panelOperation.addMouseListener(op);
            panelOperation.addMouseMotionListener(op);

          }
        }

        opUser.draw(panelOperation.getGraphics());
      }
      else if (source == userDefinedAction[4])      // Use
      {
        // User wants to go to INUSED mode

        // Check if the current mode is in DESIGN
        if (mode == DFQL.DESIGN) {
          // Clear the canvas
          panelOperation.clear();
        }

        // Retrieve stored information from the file
        String operatorName = (String)
comboUserOperators.getSelectedItem();
        // extension "udo" means User Defined Operator
        String inFile = operatorName + ".udo";

        // Save the objects in the collection
        try {
          ObjectInputStream in = new ObjectInputStream(
                                    new FileInputStream(inFile));
          opUser = (OperatorUser) in.readObject();
          in.close();
        } catch(Exception ex) {
          ex.printStackTrace();
        }

        // Set the extra information before it really can be used
        // this is in DESIGN mode
        mode = DFQL.DESIGN;
```

```
        // Set the mode of the user operator
        opUser.setMode(mode);
        // Set the drawing canvas reference
        opUser.setDesignCanvas(panelOperation);

        // Set the collection object reference
        // Current user operator vector reference that one from the file
        vUserOperators = opUser.vRefUserOperator;

        // Set the object reference
        panelOperation.refOpUser = opUser;

        // loop through each operator, and reset the some run time
properties
        Object ob = null;

        for (Enumeration en=vUserOperators.elements();
en.hasMoreElements();) {
          ob = en.nextElement();
          if (ob.getClass().getName().indexOf("Operator") != -1) {
            Operator op = (Operator) ob;
            op.refDFQLOperators = vUserOperators;
            // No need to add the mouse listener for the operaotrs
            // when the user operator is in INUSED mode
            //panelOperation.addMouseListener(op);
            //panelOperation.addMouseMotionListener(op);

          }
        }

        // Now , set to INUSED mode
        mode = DFQL.INUSED;

        // Set the mode of the user operator
        opUser.setMode(mode);
        // Set the type
        opUser.setType(opUser.getDesignName());

        opUser.draw(panelOperation.getGraphics());

        // reference to the
        panelOperation.refDFQLOperators = vDFQLOperators;

        panelOperation.addMouseListener(opUser);
        panelOperation.addMouseMotionListener(opUser);

        // Place this object into the collection object
        vDFQLOperators.add(opUser);

        // Set the operator run time collection reference
        opUser.setCollectionReference(vDFQLOperators);

      }

    }
  }
```

```
}
```

4. DFQLCanvas.java

```
/*
 * Author: Ron Chen
 * File: DFQLCanvas.java
 * A window class for DFQL Canvas for handling the graphical
 * operators and inter-relation on each other
 *
*/

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;

public class DFQLCanvas extends JPanel implements
              MouseListener, MouseMotionListener, KeyListener {
  // reference the operation collection queue
  public Vector refDFQLOperators;
  // reference to the current user operator - this works in DESIGN move
  // should be set on DFQL.java class
  public OperatorUser refOpUser = null;

  private boolean bDrag = false;

  public DFQLCanvas() {
    super();

    setBackground(Color.white);

    addMouseMotionListener(this);
    addMouseListener(this);
    addKeyListener(this);

    // Make this component capable to receive the focus
    // in this way, key events will be captured
    setRequestFocusEnabled(true);
  }

  /* --------- Mouse events -------------------- */
  public void mouseMoved(MouseEvent e) {
  }

  public void mousePressed(MouseEvent e) {
    // set the focus on this component
    requestFocus();
  }

  public void mouseReleased(MouseEvent e) {
    // System.out.println("DFQLCanvas->mouseReleased()");
    if (bDrag) {
      // redraw the objects in the collections
```

```
      // drawOperators();

      this.repaint();

      // reset to false;
      bDrag = false;
    }
  }

  public void mouseEntered(MouseEvent e) {
  }

  public void mouseExited(MouseEvent e) {
  }

  public void mouseClicked(MouseEvent e) {
    boolean bRedraw = false;


    // no need to pass event further
    e.consume();

    // if this is not the right mouse button clicked, then exit
    if ( e.getModifiers() != e.BUTTON3_MASK) {
      return;
    }

    // If this is in DESIGN mode, check the user operators
    if (refOpUser != null) {
      if (refOpUser.getMode() == DFQL.DESIGN) {
        refOpUser.mouseClicked(e);
        bRedraw = refOpUser.isDirty();
      }
    }

    for (Enumeration enum=refDFQLOperators.elements();
enum.hasMoreElements() && (!bRedraw) ;) {
      Object ob = enum.nextElement();
      if (ob.getClass().getName().indexOf("Operator") != -1) {
        Operator op = (Operator) ob;
        // pass to the event to the op
        op.mouseClicked(e);
        // obtain the value if the operator should be redrawn
        bRedraw = op.isDirty();
      }
    }

    if (bRedraw) {
      repaint();
    }
  }

  public void mouseDragged(MouseEvent e) {
    // System.out.println("DFQLCanvas->mouseDragged");
    bDrag = true;
  }

  /* ---------- End of Mouse Events --------------- */
```

```
  /* ---------- Key Events ---------------------- */
  public void keyPressed(KeyEvent e) {
    // System.out.println("DFQLCanvas->keyPressed()");
    if ((e.getKeyCode() == KeyEvent.VK_DELETE) &&
       (Operator.pickOperatorName.length() > 0)) {
        // Get the operator
        Operator op = (Operator) getObject(Operator.pickOperatorName);
        if (op != null) {
          // remove this operator from the collection list
          refDFQLOperators.removeElement(op);
        }

      // reset the focus operator name to empty string
      Operator.pickOperatorName = new String("");
      // repaint the screen
      clear();
    }
  }


  public void keyReleased(KeyEvent e) {

  }

  public void keyTyped(KeyEvent e) {

  }


  /* ---------- End of Key Events ---------------- */

  public void clear() {
     this.repaint();
  }

  public void paint(Graphics g) {
    // System.out.println("DFQLCanvas->paint()");

    super.paint(g);

    drawOperators(g);
  }


  // Redraw the objects in the collection
  private void drawOperators(Graphics g) {
    Object ob;
    //System.out.println("start DFQLCanvas->drawOperators()");
    //int i = 0;

    // Check if this is in DESIGN mode, if yes, then draw the user
operator
    if (refOpUser != null) {
      if (refOpUser.getMode() == DFQL.DESIGN) {
        refOpUser.draw(g);
      }
    }
```

```
    // Draw all the operator
    for (Enumeration e=refDFQLOperators.elements(); e.hasMoreElements()
;) {
      ob = e.nextElement();
      if (ob.getClass().getName().indexOf("Operator") != -1) {
        Operator op = (Operator) ob;

        op.draw(g);
        //System.out.println("DFQLCanvas->drawOperators(), count = " +
(++i));
      }
    }

    // Draw the input node link if this is in DESIGN mode
    if (refOpUser != null) {
      if (refOpUser.getMode() == DFQL.DESIGN) {
        refOpUser.drawLink(g);
      }
    }

    //System.out.println("end DFQLCanvas->drawOperators()");
  }

  // Get the object from the collection base the operator name
  // This is very similar routine as isInCollection()
  public Object getObject(String relation) {
      Object ob = null;
      boolean bFound = false;

      for (Enumeration e=refDFQLOperators.elements();
e.hasMoreElements() && (!bFound) ;) {
        ob = e.nextElement();
        if (ob.getClass().getName().indexOf("Operator") != -1) {
          bFound = (((Operator)
ob).operatorName).equalsIgnoreCase(relation);
        }
      }

      if (bFound)
        return ob;

      // if not found, return null
      return null;

  }


}
```

5. ExampleFileFilter.java

```
/*
 * @(#)ExampleFileFilter.java 1.8 98/08/26
 *
 * Copyright 1998 by Sun Microsystems, Inc.,
 * 901 San Antonio Road, Palo Alto, California, 94303, U.S.A.
 * All rights reserved.
 *
 * This software is the confidential and proprietary information
 * of Sun Microsystems, Inc. ("Confidential Information").  You
 * shall not disclose such Confidential Information and shall use
 * it only in accordance with the terms of the license agreement
 * you entered into with Sun.
 */


import java.io.File;
import java.util.Hashtable;
import java.util.Enumeration;
import javax.swing.*;
import javax.swing.filechooser.*;

/**
 * A convenience implementation of FileFilter that filters out
 * all files except for those type extensions that it knows about.
 *
 * Extensions are of the type ".foo", which is typically found on
 * Windows and Unix boxes, but not on Macinthosh. Case is ignored.
 *
 * Example - create a new filter that filerts out all files
 * but gif and jpg image files:
 *
 *     JFileChooser chooser = new JFileChooser();
 *     ExampleFileFilter filter = new ExampleFileFilter(
 *                   new String{"gif", "jpg"}, "JPEG & GIF Images")
 *     chooser.addChoosableFileFilter(filter);
 *     chooser.showOpenDialog(this);
 *
 * @version 1.8 08/26/98
 * @author Jeff Dinkins
 */
public class ExampleFileFilter extends FileFilter {

    private static String TYPE_UNKNOWN = "Type Unknown";
    private static String HIDDEN_FILE = "Hidden File";

    private Hashtable filters = null;
    private String description = null;
    private String fullDescription = null;
    private boolean useExtensionsInDescription = true;

    /**
     * Creates a file filter. If no filters are added, then all
     * files are accepted.
     *
     * @see #addExtension
     */
```

```
    public ExampleFileFilter() {
      this.filters = new Hashtable();
    }

    /**
     * Creates a file filter that accepts files with the given
extension.
     * Example: new ExampleFileFilter("jpg");
     *
     * @see #addExtension
     */
    public ExampleFileFilter(String extension) {
      this(extension,null);
    }

    /**
     * Creates a file filter that accepts the given file type.
     * Example: new ExampleFileFilter("jpg", "JPEG Image Images");
     *
     * Note that the "." before the extension is not needed. If
     * provided, it will be ignored.
     *
     * @see #addExtension
     */
    public ExampleFileFilter(String extension, String description) {
      this();
      if(extension!=null) addExtension(extension);
      if(description!=null) setDescription(description);
    }

    /**
     * Creates a file filter from the given string array.
     * Example: new ExampleFileFilter(String {"gif", "jpg"});
     *
     * Note that the "." before the extension is not needed adn
     * will be ignored.
     *
     * @see #addExtension
     */
    public ExampleFileFilter(String[] filters) {
      this(filters, null);
    }

    /**
     * Creates a file filter from the given string array and
description.
     * Example: new ExampleFileFilter(String {"gif", "jpg"}, "Gif and
JPG Images");
     *
     * Note that the "." before the extension is not needed and will be
ignored.
     *
     * @see #addExtension
     */
    public ExampleFileFilter(String[] filters, String description) {
      this();
      for (int i = 0; i < filters.length; i++) {
          // add filters one by one
```

```
            addExtension(filters[i]);
        }
        if(description!=null) setDescription(description);
    }

    /**
     * Return true if this file should be shown in the directory pane,
     * false if it shouldn't.
     *
     * Files that begin with "." are ignored.
     *
     * @see #getExtension
     * @see FileFilter#accepts
     */
    public boolean accept(File f) {
        if(f != null) {
            if(f.isDirectory()) {
                return true;
            }
            String extension = getExtension(f);
            if(extension != null && filters.get(getExtension(f)) != null)
{
                return true;
            };
        }
        return false;
    }

    /**
     * Return the extension portion of the file's name .
     *
     * @see #getExtension
     * @see FileFilter#accept
     */
     public String getExtension(File f) {
        if(f != null) {
            String filename = f.getName();
            int i = filename.lastIndexOf('.');
            if(i>0 && i<filename.length()-1) {
                return filename.substring(i+1).toLowerCase();
            };
        }
        return null;
    }

    /**
     * Adds a filetype "dot" extension to filter against.
     *
     * For example: the following code will create a filter that filters
     * out all files except those that end in ".jpg" and ".tif":
     *
     *   ExampleFileFilter filter = new ExampleFileFilter();
     *   filter.addExtension("jpg");
     *   filter.addExtension("tif");
     *
     * Note that the "." before the extension is not needed and will be
ignored.
     */
```

```
    public void addExtension(String extension) {
      if(filters == null) {
          filters = new Hashtable(5);
      }
      filters.put(extension.toLowerCase(), this);
      fullDescription = null;
    }


    /**
     * Returns the human readable description of this filter. For
     * example: "JPEG and GIF Image Files (*.jpg, *.gif)"
     *
     * @see setDescription
     * @see setExtensionListInDescription
     * @see isExtensionListInDescription
     * @see FileFilter#getDescription
     */
    public String getDescription() {
      if(fullDescription == null) {
          if(description == null || isExtensionListInDescription()) {
            fullDescription = description==null ? "(" : description + "
(";
            // build the description from the extension list
            Enumeration extensions = filters.keys();
            if(extensions != null) {
                fullDescription += "." + (String)
extensions.nextElement();
                while (extensions.hasMoreElements()) {
                  fullDescription += ", " + (String)
extensions.nextElement();
                }
            }
            fullDescription += ")";
          } else {
            fullDescription = description;
          }
      }
      return fullDescription;
    }

    /**
     * Sets the human readable description of this filter. For
     * example: filter.setDescription("Gif and JPG Images");
     *
     * @see setDescription
     * @see setExtensionListInDescription
     * @see isExtensionListInDescription
     */
    public void setDescription(String description) {
      this.description = description;
      fullDescription = null;
    }

    /**
     * Determines whether the extension list (.jpg, .gif, etc) should
     * show up in the human readable description.
     *
```

```
     * Only relevent if a description was provided in the constructor
     * or using setDescription();
     *
     * @see getDescription
     * @see setDescription
     * @see isExtensionListInDescription
     */
    public void setExtensionListInDescription(boolean b) {
      useExtensionsInDescription = b;
      fullDescription = null;
    }

    /**
     * Returns whether the extension list (.jpg, .gif, etc) should
     * show up in the human readable description.
     *
     * Only relevent if a description was provided in the constructor
     * or using setDescription();
     *
     * @see getDescription
     * @see setDescription
     * @see setExtensionListInDescription
     */
    public boolean isExtensionListInDescription() {
      return useExtensionsInDescription;
    }
}
```

6. FrameMain.java

```
/*
 * File: FrameMain.java
 * Written by: Ron Chen
 *
 * Last modified: Feb 10, 1999
 *
 * This is the main window for the project
 *
*/


import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import java.sql.*;
import javax.swing.*;
import javax.accessibility.*;
import javax.swing.UIManager;

public class FrameMain extends JFrame {
  // Reference the external Database object
  DB odb;

  // Object for holding the resultset data
  TableSorter sorter;
```

```
MyTableModel tableData;

//Construct the frame

// Menu Section
JMenuBar menuBar;

JMenu menuFile;
JMenuItem menuFileExit;

JMenu menuDFQL;
JMenuItem menuDFQLNew;
JMenuItem menuDFQLOpen;
JMenuItem menuDFQLSave;
JMenuItem menuDFQLRun;

JMenu menuHelp;
JMenuItem menuHelpAbout;

// Tool Bar
JToolBar toolBar;

// Panel on the center of the frame
JPanel panelCenter;

// Split Pane on the center of the panelCenter
JSplitPane splitPaneCenter;

// Split Pane on the right side of the splitPaneCenter
// for displaying the data on table format
JSplitPane splitPaneTable;

// Objects on the left side of the splitPaneCenter
JScrollPane scrollPaneDB;
JTree treeDB;

// Tab Pane on the top side of the splitPaneTable
JTabbedPane tabbedPaneProcess;

// Objects on bottom of the splitPaneTable
JTable tableQueryData;
JScrollPane scrollPaneViewQueryData;

// Panel on the first tab of the tabbedPaneProcess
JPanel panelQuery;
// Panel on the second tab of the tabbedPaneProcess
JPanel panelDFQL;
DFQL sectionDFQL;

// Objects on North of the panelQuery
JPanel panelInstruction;
JLabel labelEnterQuery;
JButton buttonExecute;
// Objects on Center of the panelQuery
JScrollPane scrollPaneEnterQuery;
JTextArea textAreaQuery;
```

```
  public static JFrame parent = new JFrame();

  public FrameMain()
  {
    parent = this;

    try
    {
      LoginDialog login = new LoginDialog(this);
      if (login.loginOption == LoginDialog.CANCEL_LOGIN) {
        login.dispose();

        System.exit(0);
      }

     // Open the database first
      odb = new DB(login.getUserName(), login.getPassword(),
                   login.getServer(), login.getDriver());

      login.dispose();

      if (odb.DBOK)
      {
        // print the table list
        // odb.PrintTableList();
        // Call the initialize function
        init();
      }
      else
      {
        System.out.println("Database can't be open");
      }
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }



  // Initialize the frame
  private void init() throws Exception
  {
    // set the layout, initial size, and title
    this.getContentPane().setLayout(new BorderLayout());
    this.setSize(new Dimension(490, 340));
    this.setTitle("Visual Database and Query");

/* Menu */
    /* File */
    menuFile = new JMenu("File");
    /* File -> Exit */
    menuFileExit = new JMenuItem("Exit");
    menuFileExit.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
```

```
    fileExit_actionPerformed(e);
  }
});
menuFile.add(menuFileExit);

/* DFQL */
menuDFQL = new JMenu("DFQL");
/* DFQL -> New */
menuDFQLNew = new JMenuItem("New");
menuDFQLNew.addActionListener(new ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    DFQLNew_actionPerformed(e);
  }
});
/* DFQL -> Open */
menuDFQLOpen = new JMenuItem("Open");
menuDFQLOpen.addActionListener(new ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    DFQLOpen_actionPerformed(e);
  }
});

/* DFQL -> Save */
menuDFQLSave = new JMenuItem("Save");
menuDFQLSave.addActionListener(new ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    DFQLSave_actionPerformed(e);
  }
});

/* DFQL -> Run */
menuDFQLRun = new JMenuItem("Run");
menuDFQLRun.addActionListener(new ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    DFQLRun_actionPerformed(e);
  }
});
menuDFQL.add(menuDFQLNew);
menuDFQL.add(menuDFQLOpen);
menuDFQL.add(menuDFQLSave);
menuDFQL.addSeparator();
menuDFQL.add(menuDFQLRun);


/* Help */
menuHelp = new JMenu("Help");
/* Help -> About */
menuHelpAbout = new JMenuItem("About");
menuHelpAbout.addActionListener(new ActionListener()
{
```

```
      public void actionPerformed(ActionEvent e)
      {
        helpAbout_actionPerformed(e);
      }
    });

    menuHelp.add(menuHelpAbout);

    menuBar = new JMenuBar();
    menuBar.add(menuFile);
    menuBar.add(menuDFQL);
    menuBar.add(menuHelp);

/* ToolBar */
    toolBar = new JToolBar();
    addTool(toolBar, "Exit");
    addTool(toolBar, "About");


/* Db Tree - Left Split Pane */
    /* Build the tree for handling the database metadata */
    // treeDB = new JTree(odb.fillDbMetaData());
    //treeDB = new JTree(new Vector());      // for testing only

    // Date: April 14, 1999
    // Using the ToolTipTree class for better data viewing
    treeDB = new ToolTipTree(odb.fillDbMetaData());

    // Make sure that the tree is scrollable
    scrollPaneDB = new JScrollPane();
    scrollPaneDB.getViewport().add(treeDB);

    // Date: April 7, 1999
    // Bug watch: JDK 1.2 (Connection class)
    // The following two lines must be placed here in order to make the
program
    // work correctly
    // What happend: When connection to the Oracle database
    //      (Personal Oracle 7.3.3) throught JDBC-ODBC driver, after
parsing
    //      the metadata on the database, and show the structure on the
Tree
    //      Style.  The connection must be closed, and re-established
again
    //      in order to make the future SQL running.
    //      Without doing this, the program will hang on
    //      statement.executeQuery() area, and going nowhere.
    //      However, this situation does not apply to Access 97
database.
    // Solution: in order to make it generic approach, apply the
following
    //           syntax to all the databases.
    // Close the connection to see if it works
    //System.out.println("Close the connection");
    odb.closeConnection();
    //System.out.println("Reopen the connection");
    odb.dbInit();
```

```
/* Right Split Pane */

/* Visual Query - Layout about controls */
    panelQuery = new JPanel(new BorderLayout());
    // 3 panels under Panel Query
    panelInstruction = new JPanel(new BorderLayout());
    scrollPaneEnterQuery = new JScrollPane();
    scrollPaneViewQueryData = new JScrollPane();

    // NORTH - for the Instruction Panel
    labelEnterQuery = new JLabel("Enter Query:");
    buttonExecute = new JButton("Execute Query");
    buttonExecute.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        buttonExecute_actionPerformed(e);
      }
    });
    panelInstruction.setLayout(new BorderLayout());
    panelInstruction.add(labelEnterQuery, BorderLayout.WEST);
    panelInstruction.add(buttonExecute, BorderLayout.EAST);

    // CENTER - for the Enter Query Panel
    textAreaQuery = new JTextArea();
    scrollPaneEnterQuery.getViewport().add(textAreaQuery, null);

    // Place in the Query Panel
    panelQuery.add(panelInstruction, BorderLayout.NORTH);
    panelQuery.add(scrollPaneEnterQuery, BorderLayout.CENTER);


/* DFQL */
    sectionDFQL = new DFQL();
    panelDFQL = new JPanel(new BorderLayout());
    panelDFQL.add(sectionDFQL, BorderLayout.CENTER);


    /* Tabs layout */
    tabbedPaneProcess = new JTabbedPane();
    tabbedPaneProcess.addTab("Query", panelQuery);
    tabbedPaneProcess.addTab("DFQL", panelDFQL);
    tabbedPaneProcess.setTabPlacement(JTabbedPane.BOTTOM);

    // Bottom - for the View Query Data Panel
    // Create the table to handle the resultset
    tableData = new MyTableModel();
    sorter = new TableSorter(tableData);
    tableQueryData = new JTable(sorter);

    // Install a mouse listener in the TableHeader as the sorter UI.
    sorter.addMouseListenerToHeaderInTable(tableQueryData);

    // scrollPaneViewQueryData =
JTable.createScrollPaneForTable(tableQueryData);
    scrollPaneViewQueryData = new JScrollPane(tableQueryData);
```

```
    /* Db Tree - Left Split Pane */
    scrollPaneDB.getViewport().add(treeDB, BorderLayout.CENTER);

/* Split Pane - Vertical Split */
/* Top Component    - tabPaneProcess
   Bottom Component - scrollPaneViewQueryData
*/
    splitPaneTable = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
                                    tabbedPaneProcess,
                                    scrollPaneViewQueryData);
    splitPaneTable.setOneTouchExpandable(true);
    //splitPaneTable.setMinimumSize(new Dimension(100,100));

/* Split Pane */
/* Left Component - Db Tree
   Right Component - splitPaneTable
*/
    splitPaneCenter = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                                     scrollPaneDB,
                                     splitPaneTable);
    splitPaneCenter.setOneTouchExpandable(true);
    // splitPaneCenter.setMinimumSize(new Dimension(100,100));

/* Place the splitpane in the center of the panel */
    panelCenter = new JPanel(new BorderLayout());
    panelCenter.add(splitPaneCenter, BorderLayout.CENTER);

    this.setJMenuBar(menuBar);
    this.getContentPane().add(toolBar, BorderLayout.NORTH);
    this.getContentPane().add(panelCenter, BorderLayout.CENTER);


    // Valide the frame
    this.validate();

    //Maximize the window
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = this.getSize();
    if (frameSize.height > screenSize.height)
      frameSize.height = screenSize.height;
    if (frameSize.width > screenSize.width)
      frameSize.width = screenSize.width;
    // Center the windows
    // frame.setLocation((screenSize.width - frameSize.width) / 2,
(screenSize.height - frameSize.height) / 2);


    // Maximize the windows
    this.setSize(screenSize.width, screenSize.height);
    this.setVisible(true);


  }


  //File | Exit action performed
  public void fileExit_actionPerformed(ActionEvent e)
  {
```

```
    System.exit(0);
  }

  //DFQL | New action performed
  public void DFQLNew_actionPerformed(ActionEvent e){
    sectionDFQL.newDFQL();
  }

  //DFQL | Open action performed
  public void DFQLOpen_actionPerformed(ActionEvent e){
    sectionDFQL.openDFQL();
  }

  //DFQL | Save action performed
  public void DFQLSave_actionPerformed(ActionEvent e){
    sectionDFQL.saveDFQL();
  }

  //DFQL | Run action performed
  public void DFQLRun_actionPerformed(ActionEvent e)
  {
    String query = sectionDFQL.getQuery();
    System.out.println("Run DFQL query: " + query);

    if (query.length() == 0) {
      System.out.println("No query to run");
      return;
    }

    // Now, run the query
    executeUserQuery(query);

  }

  //Help | About action performed
  public void helpAbout_actionPerformed(ActionEvent e)
  {
    AboutBox dlg = new AboutBox(this);
    Dimension dlgSize = dlg.getPreferredSize();
    Dimension frmSize = getSize();
    Point loc = getLocation();
    dlg.setLocation((frmSize.width - dlgSize.width) / 2 + loc.x,
(frmSize.height - dlgSize.height) / 2 + loc.y);
    dlg.setModal(true);
    dlg.show();
  }
  // Add a button to the ToolBar
  public void addTool(JToolBar toolBar, String name) {
        JButton b = (JButton) toolBar.add(
              new JButton(loadImageIcon("images/" + name +
".gif",name)));
        b.setToolTipText(name);
        b.setMargin(new Insets(0,0,0,0));
        b.setActionCommand(name);
        b.getAccessibleContext().setAccessibleName(name);
        b.addActionListener(new ActionListener()
        {
```

```
            public void actionPerformed(ActionEvent e)
            {
              if (e.getActionCommand().equals("Exit")) {
                fileExit_actionPerformed(e);
              }
              else if (e.getActionCommand().equals("About")) {
                System.out.println("Click <About> tool bar");
                helpAbout_actionPerformed(e);
              }
            }
          });

  }

  public void buttonExecute_actionPerformed(ActionEvent e)
  {
    System.out.println("Execute the query");

    String sQuery = textAreaQuery.getText();
    System.out.println(sQuery);

    // run the query
    executeUserQuery(sQuery);


  }


  // execute the user define query
  public void executeUserQuery(String sQuery) {
    try {
      if (odb.executeQuery(sQuery) == true)
      {
        // System.out.println("return true after executing the query");
        // System.out.println("First column name " +
odb.columnNames[1]);

        // System.out.println("First row and column data " +
((Vector)odb.rows.elementAt(1)).elementAt(1));

        // repaint the table data
        tableData.refreshTable(odb.rows, odb.columnNames, odb.resultSet,
odb.metaData);

      }
      else
        System.out.println("return false after executing the query");
    } catch (Exception err) {
      System.out.println(err);
    }

  }

  public ImageIcon loadImageIcon(String filename, String description) {
      return new ImageIcon(filename, description);
  }
```

```
  public static void main(String[] args)
  {
    try
    {
      UIManager.setLookAndFeel(new
com.sun.java.swing.plaf.windows.WindowsLookAndFeel());
      //UIManager.setLookAndFeel(new
com.sun.java.swing.plaf.motif.MotifLookAndFeel());
      //UIManager.setLookAndFeel(new
com.sun.java.swing.plaf.metal.MetalLookAndFeel());
    }
    catch (Exception e)
    {
        System.err.println(e);
    }

    new FrameMain();

  }
}
```

7. InputBarNode.java

```
/*
 * Author: Ron Chen
 * File: InputBarNode.java
 *
 * A Class that uses as helper class to record the input node
 * information for the OperatorUser class
 *
 *
*/

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import java.io.*;
import java.util.*;

public class InputBarNode implements Externalizable {

  // 3 type of inputbar node
  public final static int RELATION = 1;
  public final static int CONDITION = 2;
  public final static int ATTRIBUTE = 3;

  /* Reference to the collection list of the user operator list */
  public Vector vRefUserOperator = null;
  public Vector vRefDFQLOperator = null;

  /* property */
  // Location the of the node
  public int ox = 0;
  public int oy = 0;
```

```
  // Node type
  public int nodeType = 0;
  // Node seqence: 1, 2, 3, ...
  public int sequence = 0;


  // Input value - for INUSED mode
  String inputValue = "";

  // Which operator is connected to
  String targetOperatorName = "";
  // Which node is linked to: 1, 2, 3, ...
  int targetOperatorNode = 0;

  /* end of property */

  String[] labels = {"Link to Target Operator Name", "Target Operator
Node (1,2,...)"};
  JLabel[] propertyLabels = new JLabel[labels.length];
  JTextField[] propertyTextFields = new JTextField[labels.length];


  public InputBarNode() {
    // empty body
  }

  public InputBarNode(int seq) {
    sequence = seq;
  }

  public void setX(int x) {
    ox = x;
  }

  public int getX() {
    return ox;
  }

  public void setY(int y) {
    oy = y;
  }

  public int getY() {
    return oy;
  }

  public void setNodeType(int type) {
    nodeType = type;
  }

  public int getNodeType() {
    return nodeType;
  }

  public void setSequence(int i) {
    sequence = i;
  }
```

```
  public int getSequence() {
    return sequence;
  }

  public void setInputValue(String s) {
    inputValue = new String(s);
  }

  public String getInputValue() {
    return inputValue;
  }

  // Set the property on this node through the dialog
  public void setProperty() {
    for (int i=0; i<labels.length; ++i) {
        String fieldText="";

        // Assigned the filed name
        propertyLabels[i] = new JLabel(labels[i]);
        // Get the filed information
        switch (i) {
          case 0:           // target operator name
            fieldText = new String(targetOperatorName);
            break;
          case 1:           // target operator node number
            fieldText = String.valueOf(targetOperatorNode);
            break;
          default:
        }
        propertyTextFields[i] = new JTextField(new String(fieldText));
     }

    // System.out.println("popup the property window");

    PropertyWindow propertyWindow = new PropertyWindow(FrameMain.parent,
                                                       propertyLabels,
propertyTextFields);

    if (propertyWindow.propertyOption == propertyWindow.OK) {
      // Save the changes
      for (int i=0; i<labels.length; ++i) {

        String fieldText = propertyTextFields[i].getText();

        // Get the field inforamtion from the text field
        switch (i) {
          case 0:           // target operator name
            targetOperatorName = new String(fieldText);
            break;
          case 1:           // target operator node
            targetOperatorNode = (new Integer(fieldText)).intValue();
            break;
          default:
        }

      }
```

```
    }
    propertyWindow.dispose();

  }


  // Implement Externalizable interface
  public void writeExternal(ObjectOutput out) throws IOException {
    // Write the data only
    out.writeInt(ox);                    out.writeInt(oy);
    out.writeInt(nodeType);
    out.writeInt(sequence);
    out.writeObject(inputValue);
    out.writeObject(targetOperatorName);
    out.writeInt(targetOperatorNode);
  }


  // Implement Externalizable interface
  public void readExternal(ObjectInput in) {
    // save the data
    try {
      ox = in.readInt();         oy = in.readInt();
      nodeType = in.readInt();
      sequence = in.readInt();
      inputValue = (String) in.readObject();
      targetOperatorName = (String) in.readObject();
      targetOperatorNode = in.readInt();
   } catch (Exception e) {
      System.err.println(e);
   }
  }

}
```

8. LoginDialog.java

```
/*
 * File: LoginDialog.java
 *
 * Written by: Ron Chen
 * Date: Feb 14, 1999
 *
 * This is part of thesis project for MSCS in NPS
*/

/**
 * A a login dialog to allow user to input the login information
 * to the database
 *
*/

import java.awt.*;
import javax.swing.*;
```

```
import java.awt.event.*;

public class LoginDialog extends JDialog {
  static String[] options = { "OK", "CANCEL" };
  static String   loginTitle = "Connection Information";

  public static final int LOGIN_NOW = 1;
  public static final int CANCEL_LOGIN = 2;

  JLabel      userNameLabel;
  JTextField  userNameField;
  JLabel      passwordLabel;
  JTextField  passwordField;
  JLabel      serverLabel;
  JTextField  serverField;
  JLabel      driverLabel;
  JTextField  driverField;

  JButton     loginButton;
  JButton     cancelButton;

  JPanel      loginPanel;

  public int loginOption;

  public LoginDialog(JFrame owner) {

    // Create the labels and text fields.
    userNameLabel = new JLabel("User name: ", JLabel.RIGHT);
        userNameField = new JTextField("");

        passwordLabel = new JLabel("Password: ", JLabel.RIGHT);
        passwordField = new JTextField("");

    serverLabel = new JLabel("Database URL: ", JLabel.RIGHT);
        serverField = new JTextField("jdbc:odbc:???");

        driverLabel = new JLabel("Driver: ", JLabel.RIGHT);
        driverField = new JTextField("sun.jdbc.odbc.JdbcOdbcDriver");


        loginPanel = new JPanel(false);
        loginPanel.setLayout(new BoxLayout(loginPanel,
                                    BoxLayout.X_AXIS));

        JPanel namePanel = new JPanel(false);
        namePanel.setLayout(new GridLayout(0, 1));
        namePanel.add(userNameLabel);
        namePanel.add(passwordLabel);
        namePanel.add(serverLabel);
        namePanel.add(driverLabel);

      JPanel fieldPanel = new JPanel(false);
      fieldPanel.setLayout(new GridLayout(0, 1));
      fieldPanel.add(userNameField);
      fieldPanel.add(passwordField);
      fieldPanel.add(serverField);
    fieldPanel.add(driverField);
```

```
      // Create the buttons.
/*    loginButton = new JButton("Login");
    loginButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            loginOption = LOGIN_NOW;
          }
        }
      );

      cancelButton = new JButton("Cancel");
      loginButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            loginOption = CANCEL_LOGIN;
          }
        }
      );

      JPanel buttonPanel = new JPanel(false);
    buttonPanel.setLayout(new GridLayout(0, 1));
    buttonPanel.add(loginButton);
    buttonPanel.add(cancelButton);
*/

    // Place to the loginPanel
      loginPanel.add(namePanel);
      loginPanel.add(fieldPanel);
//    loginPanel.add(buttonPanel);

    int nOption = JOptionPane.showOptionDialog(owner, loginPanel,
loginTitle,

JOptionPane.DEFAULT_OPTION,

JOptionPane.INFORMATION_MESSAGE,
                                                null, options,
options[0]);

    System.out.println("Login option = " + nOption);

    switch (nOption)
    {
      case 0:
        loginOption = LOGIN_NOW;
        break;
      case 1:
        loginOption = CANCEL_LOGIN;
        break;
      case JOptionPane.CLOSED_OPTION:
        loginOption = CANCEL_LOGIN;
    }


   }

  public String getUserName() {
      return new String(userNameField.getText());
```

```
  }

  public String getPassword() {
    return new String(passwordField.getText());
  }

  public String getServer() {
    return new String(serverField.getText());
  }

  public String getDriver() {
    return new String(driverField.getText());
  }

}
```

9. MultiLineToolTip.java

```
/*
 * Author: Ron Chen
 * File: MultiLineToolTipUI.java
 *
 * Wrapper Class for handling mulitiline ToolTip User Interface
 *
 * Credit: Most of the code are download from Sun's JFC Conference
 *         Forum.
*/
import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import javax.swing.plaf.metal.*;
import javax.swing.plaf.*;

public class MultiLineToolTip extends JToolTip {

  public MultiLineToolTip() {
      setUI(new MultiLineToolTipUI());
  }

  // Inner Class - User Interface for multi-line ToolTip
  public class MultiLineToolTipUI extends ToolTipUI {
    private String[] strs;
    private int maxWidth = 0;

    public void paint(Graphics g, JComponent c) {
      if (strs == null) {
        c.setVisible(false);
        return;
      }

      // make sure the component is visible
      c.setVisible(true);
```

```
      FontMetrics metrics =
Toolkit.getDefaultToolkit().getFontMetrics(g.getFont());
      Dimension size = c.getSize();
      // g.setColor(c.getBackground());
      g.setColor(Color.yellow);
      g.fillRect(1, 1, size.width-1, size.height-1);
      g.setColor(Color.black);
      g.drawRect(0, 0, size.width, size.height);
      g.setColor(c.getForeground());
      if (strs != null) {
        for (int i=0;i<strs.length;i++) {
            g.drawString(strs[i], 3, (metrics.getHeight()) * (i+1));
        }
      }
    }

    public Dimension getPreferredSize(JComponent c) {
      FontMetrics metrics =
Toolkit.getDefaultToolkit().getFontMetrics(c.getFont());
      String tipText = ((JToolTip)c).getTipText();
      if (tipText == null) {
        strs = null;
        tipText = "";
        return (new Dimension(0,0));
      }
      BufferedReader br = new BufferedReader(new StringReader(tipText));
      String line;
      int maxWidth = 0;
      Vector v = new Vector();
      try {
        while ((line = br.readLine()) != null) {
            int width = SwingUtilities.computeStringWidth(metrics,line);
            maxWidth = (maxWidth < width) ? width : maxWidth;
            v.addElement(line);
        }
      } catch (IOException ex) {
        ex.printStackTrace();
      }
      int lines = v.size();
      if (lines < 1) {
        strs = null;
        lines = 1;
      } else {
        strs = new String[lines];
        int i=0;
        for (Enumeration e = v.elements(); e.hasMoreElements() ;i++) {
          strs[i] = (String)e.nextElement();
        }
      }
      int height = metrics.getHeight() * lines;
      this.maxWidth = maxWidth;
      return new Dimension(maxWidth + 6, height + 4);
    }
  }

}
```

10. MyTableModel.java

```
/*
 * File: MyTableModel.java
 * Written by: Ron Chen
 *
 * Part of the source codes are from the JDK sample codes, and modified
for
 * this project
 *
 * A class displays the data to the table format (rows and columns)
 *
*/

import java.util.Vector;
import java.sql.*;
import javax.swing.table.AbstractTableModel;
import javax.swing.event.TableModelEvent;

public class MyTableModel extends AbstractTableModel {
    Connection          connection;
    Statement           statement;
    ResultSet           resultSet;
    String[]            columnNames;
    Class[]             columnTpyes;
    Vector                  rows;
    ResultSetMetaData   metaData;

    public MyTableModel() {
      rows = new Vector();
      columnNames = new String[]{};
    }

    public void refreshTable(Vector data, String[] colNames, ResultSet
rs, ResultSetMetaData rsmd)
    {
      rows = data;
      columnNames = colNames;
      resultSet = rs;
      metaData = rsmd;

      fireTableChanged(null); // Tell the listeners a new table has
arrived.
    }


//////////////////////////////////////////////////////////////////////
//
    //
    //              Implementation of the TableModel Interface
    //

//////////////////////////////////////////////////////////////////////
//

    // MetaData

    public String getColumnName(int column) {
```

```
        if (columnNames[column] != null) {
            return columnNames[column];
        } else {
            return "";
        }
    }

    public Class getColumnClass(int column) {
        int type;
        try {
            type = metaData.getColumnType(column+1);
        }
        catch (SQLException e) {
            return super.getColumnClass(column);
        }

        switch(type) {
        case Types.CHAR:
        case Types.VARCHAR:
        case Types.LONGVARCHAR:
            return String.class;

        case Types.BIT:
            return Boolean.class;

        case Types.TINYINT:
        case Types.SMALLINT:
        case Types.INTEGER:
            return Integer.class;

        case Types.BIGINT:
            return Long.class;

        case Types.FLOAT:
        case Types.DOUBLE:
            return Double.class;

        case Types.DATE:
            return java.sql.Date.class;

        default:
            return Object.class;
        }
    }

    public boolean isCellEditable(int row, int column) {
        try {
            return metaData.isWritable(column+1);
        }
        catch (SQLException e) {
            return false;
        }
    }

    public int getColumnCount() {
        return columnNames.length;
    }
```

```
    // Data methods

    public int getRowCount() {
        return rows.size();
    }

    public Object getValueAt(int aRow, int aColumn) {
        Vector row = (Vector)rows.elementAt(aRow);
        return row.elementAt(aColumn);
    }

    public String dbRepresentation(int column, Object value) {
        int type;

        if (value == null) {
            return "null";
        }

        try {
            type = metaData.getColumnType(column+1);
        }
        catch (SQLException e) {
            return value.toString();
        }

        switch(type) {
        case Types.INTEGER:
        case Types.DOUBLE:
        case Types.FLOAT:
            return value.toString();
        case Types.BIT:
            return ((Boolean)value).booleanValue() ? "1" : "0";
        case Types.DATE:
            return value.toString(); // This will need some conversion.
        default:
            return "\""+value.toString()+"\"";
        }

    }

    public void setValueAt(Object value, int row, int column) {
        try {
            String tableName = metaData.getTableName(column+1);
            // Some of the drivers seem buggy, tableName should not be
null.
            if (tableName == null) {
                System.out.println("Table name returned null.");
            }
            String columnName = getColumnName(column);
            String query =
                "update "+tableName+
                " set "+columnName+" = "+dbRepresentation(column,
value)+
                " where ";
            // We don't have a model of the schema so we don't know the
            // primary keys or which columns to lock on. To demonstrate
            // that editing is possible, we'll just lock on everything.
            for(int col = 0; col<getColumnCount(); col++) {
```

```
                String colName = getColumnName(col);
                if (colName.equals("")) {
                    continue;
                }
                if (col != 0) {
                    query = query + " and ";
                }
                query = query + colName +" = "+
                    dbRepresentation(col, getValueAt(row, col));
            }
            System.out.println(query);
            System.out.println("Not sending update to database");
            // statement.executeQuery(query);
        }
        catch (SQLException e) {
            //     e.printStackTrace();
            System.err.println("Update failed");
        }
        Vector dataRow = (Vector)rows.elementAt(row);
        dataRow.setElementAt(value, column);

    }
}
```

11. Operator.java

```
/*
 * Author: Ron Chen
 * File: Operator.java
 *
 * Last Modified: March 17, 1999
 *
 * A base class for variety of DFQL operators
 *
*/

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import java.io.*;
import java.util.*;

public class Operator implements MouseListener, MouseMotionListener,
Externalizable
{

  public static int NODE_RADIUS = 8;

  // reference the operation collection queue
  // Don't make this as static variable, this can be used on
  // many different Vector collections
  // eg. regular DFQLOperator Vector, or user defined Operator Vector
  public Vector refDFQLOperators=null;
```

```
// Position of each node (initialize)
// Input Node 1
public double x1 = 0;
public double y1 = 0;

// Input Node 2
public double x2 = 0;
public double y2 = 0;

// Input Node 3
public double x3 = 0;
public double y3 = 0;

// Input Node 4
public double x4 = 0;
public double y4 = 0;

// output node
public double ox = 0;
public double oy = 0;

// What is the focus operator name
public static String pickOperatorName="";

// Type: select, project, ....
String operatorType = "";

// -------- common property ---------------------
// Name: whatever that user names it
String operatorName = "";

// Position of the operator (main body)
double x = 10;
double y = 10;

// operator (height and width of the main body)
double height = 20;
double width = 80;
// -------- end of property ---------------


// boolean value if this is mouseDraged event
boolean bDrag = false;

// boolean value if this is mouseClicked event
boolean bPick = false;

// boolean value if this operator should be redraw
boolean bRedraw = false;

public Operator() {
  setX(40);
  setY(40);

}


public Operator(String type) {
```

```
    this();
    setType(type);
  }

  public Operator(String type, Vector vRef) {
    this(type);
    refDFQLOperators = vRef;

    // since the reference is available
    // then set the default operator name for this object
    setDefaultName();
  }


  public String toString() {
    StringBuffer s = new StringBuffer();
    s.append(getClass().toString());
    s.append(" ObjectName = " + operatorName);
    if (x1 != 0 || y1 != 0) {
      s.append(" (x1, y1) = [" + x1 + ", " + y1 + "]");
    }
    if (x2 != 0 || y2 != 0) {
      s.append(" (x2, y2) = [" + x2 + ", " + y2 + "]");
    }
    if (x3 != 0 || y3 != 0) {
      s.append(" (x3, y3) = [" + x3 + ", " + y3 + "]");
    }
    if (x4 != 0 || y4 != 0) {
      s.append(" (x4, y4) = [" + x4 + ", " + y4 + "]");
    }
    if (ox != 0 || oy != 0) {
      s.append(" (ox, oy) = [" + ox + ", " + oy + "]");
    }

    return s.toString();

  }


  public void setType(String type) {
    operatorType = new String(type);
  }

  public String getType() {
    return new String(operatorType);
  }

  public void setName(String name) {
    operatorName = new String(name);
  }

  public String getName() {
    return new String(operatorName);
  }

  public void setX(double posX) {
    x = posX;
  }
```

```
public double getX() {
  return x;
}

public void setY(double posY) {
  y = posY;
}

public double getY() {
  return y;
}

public void setHeight(double h) {
  height = h;
}

public double getHeight() {
  return height;
}

public void setWidth(double w) {
  width = w;
}

public double getWidth() {
  return width;
}

public void setCollectionReference(Vector vRef) {
  refDFQLOperators = vRef;

  // Check if the Operator Name is set, if not, then set one
  if (operatorName.length() == 0) {
    setDefaultName();
  }
}

// determine if the operator should be redraw
public boolean isDirty() {
  return bRedraw;
}

// set/reset the redraw value
public void setDirty(boolean value) {
  bRedraw = value;
}

// This method is specific used by OperatorUsre.java class
// when the input node node link to the InputBar Node
// when the buildQuery() method, the value of each link node
// will pass to the current operator.  Once the query is built,
// the input node value will be reset to the original value
public void setInputNodeValue(int nNode, String value) {
  // empty body - should extended by each child class
}

public void setDefaultName() {
```

```
  // Set the default operator name = operatorxx
  if (refDFQLOperators != null) {
     int n = refDFQLOperators.size();
     setName("Operator" + (++n));
  }

}

/* --------- Mouse events -------------------- */
public void mouseDragged(MouseEvent e) {
  //e.consume();
  if (bPick) {
    bDrag = true;
  }
  // System.out.println("Operator->mouseDragged(), bDrag = " + bDrag);

}

public void mouseMoved(MouseEvent e) {
}

public void mousePressed(MouseEvent e) {
  // Determine if the events in the defined area
  // and left button is clicked
  if (isInBound(e.getX(), e.getY()) &&
      (e.getModifiers() == e.BUTTON1_MASK)) {
    pickOperatorName = operatorName;

    bPick = true;

    // System.out.println("Operator->mousePressed() in bound");
  }
  else {
    bPick = false;

    // System.out.println("Operator->mousePressed() out bound");
  }
}

public void mouseReleased(MouseEvent e) {
  //System.out.println("Operator->mouseReleased()");
  //e.consume();
  if (bDrag) {
    // Draw the object again
    x = e.getX();
    y = e.getY();

    // reset the value
    bDrag = false;

  }

  // release the holding if the previous events is mousePressed();
  bPick = false;
}

public void mouseEntered(MouseEvent e) {
}
```

```
  public void mouseExited(MouseEvent e) {

  }

  public void mouseClicked(MouseEvent e) {

  }

  /* ---------- End of Mouse Events --------------- */


  // Draw the main body
  public void draw(Graphics graphics) {

    if (graphics == null) {
      System.out.println("graphics is null on Operator->draw()");
      return;
    }


    // determine the width and height of the body
    // by calculating the width and height of the string on the current
font
    FontMetrics fm = graphics.getFontMetrics();
    width  = fm.stringWidth(operatorType) + 20;
    height = fm.getHeight() + 10;


    //System.out.println("Set the graphics reference");
    Graphics2D g = (Graphics2D) graphics;
    //System.out.println("create the rectangle object");
    Shape shape = new RoundRectangle2D.Double(x, y, width, height, 8,
8);
    //System.out.println("draw the shape");
    g.draw(shape);

    // draw the operator type name
    g.drawString(operatorType,(int)(x + 10), (int)(y + height/2 + 5));
  }


  // Draw two input nodes and one output node
  public void drawTwoInputNodes(Graphics graphics) {
    if (graphics == null) {
      System.out.println("graphics is null on Operator-
>drawTwoInputNodes()");
      return;
    }

    // paint(graphics);
    //System.out.println("Set the graphics reference");
    Graphics2D g = (Graphics2D) graphics;
    //System.out.println("draw the Input Node 1");
    x1 = x + (getWidth()/4)- (NODE_RADIUS/2);
    y1 = y - NODE_RADIUS;
```

```
    Shape inputNode1 = new
Ellipse2D.Double(x1,y1,NODE_RADIUS,NODE_RADIUS);
    g.draw(inputNode1);

    //System.out.println("draw the Input Node 2");
    x2 = x + (getWidth()/4)*3 -(NODE_RADIUS/2);
    y2 = y1;
    Shape inputNode2 = new
Ellipse2D.Double(x2,y2,NODE_RADIUS,NODE_RADIUS);
    //System.out.println("draw the Input Node 2");
    g.draw(inputNode2);

    //System.out.println("draw the Output Node");
    ox = x + getWidth()/2 - (NODE_RADIUS/2);
    oy = y + getHeight();
    Shape outputNode= new
Ellipse2D.Double(ox,oy,NODE_RADIUS,NODE_RADIUS);
    g.draw(outputNode);

  }


  // Draw three input nodes and one output node
  public void drawThreeInputNodes(Graphics graphics) {
    if (graphics == null) {
      System.out.println("graphics is null on Operator-
>drawThreeInputNodes()");
      return;
    }

    // paint(graphics);
    //System.out.println("Set the graphics reference");
    Graphics2D g = (Graphics2D) graphics;
    //System.out.println("draw the Input Node 1");
    x1 = x + (getWidth()/4)- (NODE_RADIUS/2);
    y1 = y - NODE_RADIUS;
    Shape inputNode1 = new
Ellipse2D.Double(x1,y1,NODE_RADIUS,NODE_RADIUS);
    g.draw(inputNode1);

    //System.out.println("draw the Input Node 2");
    x2 = x + getWidth()/2 - (NODE_RADIUS/2);
    y2 = y1;
    Shape inputNode2 = new
Ellipse2D.Double(x2,y2,NODE_RADIUS,NODE_RADIUS);
    //System.out.println("draw the Input Node 2");
    g.draw(inputNode2);


    //System.out.println("draw the Input Node 3");
    x3 = x + (getWidth()/4)*3 -(NODE_RADIUS/2);
    y3 = y1;
    Shape inputNode3 = new
Ellipse2D.Double(x3,y3,NODE_RADIUS,NODE_RADIUS);
    //System.out.println("draw the Input Node 3");
    g.draw(inputNode3);
```

```
    //System.out.println("draw the Output Node");
    ox = x + getWidth()/2 - (NODE_RADIUS/2);
    oy = y + getHeight();
    Shape outputNode= new
Ellipse2D.Double(ox,oy,NODE_RADIUS,NODE_RADIUS);
    g.draw(outputNode);

  }


  // Draw four input nodes and one output node
  public void drawFourInputNodes(Graphics graphics) {
    if (graphics == null) {
      System.out.println("graphics is null on Operator-
>drawFourInputNodes()");
      return;
    }

    // paint(graphics);
    //System.out.println("Set the graphics reference");
    Graphics2D g = (Graphics2D) graphics;
    //System.out.println("draw the Input Node 1");
    x1 = x + (getWidth()/5)- (NODE_RADIUS/2);
    y1 = y - NODE_RADIUS;
    Shape inputNode1 = new
Ellipse2D.Double(x1,y1,NODE_RADIUS,NODE_RADIUS);
    g.draw(inputNode1);

    //System.out.println("draw the Input Node 2");
    x2 = x + (getWidth()/5)*2 - (NODE_RADIUS/2);
    y2 = y1;
    Shape inputNode2 = new
Ellipse2D.Double(x2,y2,NODE_RADIUS,NODE_RADIUS);
    //System.out.println("draw the Input Node 2");
    g.draw(inputNode2);


    //System.out.println("draw the Input Node 3");
    x3 = x + (getWidth()/5)*3 -(NODE_RADIUS/2);
    y3 = y1;
    Shape inputNode3 = new
Ellipse2D.Double(x3,y3,NODE_RADIUS,NODE_RADIUS);
    //System.out.println("draw the Input Node 3");
    g.draw(inputNode3);

    //System.out.println("draw the Input Node 4");
    x4 = x + (getWidth()/5)*4 -(NODE_RADIUS/2);
    y4 = y1;
    Shape inputNode4 = new
Ellipse2D.Double(x4,y4,NODE_RADIUS,NODE_RADIUS);
    //System.out.println("draw the Input Node 4");
    g.draw(inputNode4);

    //System.out.println("draw the Output Node");
    ox = x + getWidth()/2 - (NODE_RADIUS/2);
    oy = y + getHeight();
    Shape outputNode= new
Ellipse2D.Double(ox,oy,NODE_RADIUS,NODE_RADIUS);
```

```
    g.draw(outputNode);

  }


  // Determinte if this is inbound
  protected boolean isInBound(int cx, int cy) {
    if ((cx >= x) && (cx <= (x+width)) &&
        (cy >= y) && (cy <= (y+height))) {
      return true;
    }
    return false;
  }

  // Determine if the current operator name is in the collection object
  public boolean isInCollection(String relation) {
    boolean bFound = false;
    Object ob;

    for (Enumeration e=refDFQLOperators.elements(); e.hasMoreElements()
&& (!bFound) ;) {
      ob = e.nextElement();
      if (ob.getClass().getName().indexOf("Operator") != -1) {
        Operator op = (Operator) ob;
        bFound = (op.operatorName).equalsIgnoreCase(relation);
      }
    }

    return bFound;
  }

  // Get the object from the collection base the operator name
  // This is very similar routine as isInCollection()
  public Object getObject(String relation) {
      Object ob = null;
      boolean bFound = false;

      for (Enumeration e=refDFQLOperators.elements();
e.hasMoreElements() && (!bFound) ;) {
        ob = e.nextElement();
        if (ob.getClass().getName().indexOf("Operator") != -1) {
          bFound = (((Operator)
ob).operatorName).equalsIgnoreCase(relation);
        }
      }

      if (bFound)
        return ob;

      // if not found, return null
      return null;

  }

  // Action area
  public String buildQuery() {
    return (new String(""));
  }
```

```
  // Implement Externalizable interface
  public void writeExternal(ObjectOutput out) throws IOException {
    // Write the data only
    out.writeObject(operatorType);      out.writeObject(operatorName);
    out.writeDouble(x);                 out.writeDouble(y);
    out.writeDouble(width);             out.writeDouble(height);
    out.writeDouble(x1);                out.writeDouble(y1);
    out.writeDouble(x2);                out.writeDouble(y2);
    out.writeDouble(x3);                out.writeDouble(y3);
    out.writeDouble(x4);                out.writeDouble(y4);
    out.writeDouble(ox);                out.writeDouble(oy);
    out.writeBoolean(bPick);
    out.writeBoolean(bDrag);
    out.writeBoolean(bRedraw);
  }


  // Implement Externalizable interface
  public void readExternal(ObjectInput in) {
    // save the data
    try {
      operatorType = (String) in.readObject();
      operatorName = (String) in.readObject();
      x  = in.readDouble();      y  = in.readDouble();
      width = in.readDouble();   height = in.readDouble();
      x1 = in.readDouble();      y1 = in.readDouble();
      x2 = in.readDouble();      y2 = in.readDouble();
      x3 = in.readDouble();      y3 = in.readDouble();
      x4 = in.readDouble();      y4 = in.readDouble();
      ox = in.readDouble();      oy = in.readDouble();
      bPick = in.readBoolean();
      bDrag = in.readBoolean();
      bRedraw = in.readBoolean();

      /*
      System.out.print("OperatorType = " + operatorType +
                       " OperatorName = " + operatorName + "\n" +
                       "x, y = " + x + ", " + y);
      */

    } catch (Exception e) {
      System.err.println(e);
    }
  }


}
```

12. OperatorDiff.java

```
/*
 * Author: Ron Chen
 * File: OperatorDiff.java
```

```
 *
 * A Class for diff operator for DFQL
 *
*/

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import java.io.*;
import java.util.*;


public class OperatorDiff extends Operator
{

  final static String OPERATOR_TYPE = "diff";

  String[] labels = {"Name", "Position X", "Position Y",
                     "Relation 1", "Relation 2"};
  JLabel[] propertyLabels = new JLabel[labels.length];
  JTextField[] propertyTextFields = new JTextField[labels.length];

  // ------ Property -----------------
  private String relation1="";
  private String relation2="";

  // ------ end of property ----------

  public OperatorDiff() {
    super(OPERATOR_TYPE);
  }

  public OperatorDiff(Vector vRef) {
    super(OPERATOR_TYPE, vRef);
  }

  // -------- Property get/set ------------
  public String getRelation1() {
    return relation1;
  }

  public void setRelation1(String sRelation) {
    relation1 = new String(sRelation);
  }

  public String getRelation2() {
    return relation2;
  }

  public void setRelation2(String sRelation) {
    relation2 = new String(sRelation);
  }

  // --------- end of property get/set -------

  // This method is specific used by OperatorUsre.java class
  // when the input node node link to the InputBar Node
```

```
// when the buildQuery() method, the value of each link node
// will pass to the current operator.  Once the query is built,
// the input node value will be reset to the original value
public void setInputNodeValue(int nNode, String value) {
  // empty body - should extended by each child class
  switch (nNode) {
    case 1:         // relation 1
      setRelation1(value);
      break;
    case 2:         // relation 2
      setRelation2(value);
      break;
    default:
  }
}
// mouse events on this class
public void mouseClicked(MouseEvent e) {
  // call the base class event
  super.mouseClicked(e);

  // if this is right mouse clicked, then popup
  // property window
  if (!isInBound(e.getX(), e.getY())) {
    return ;
  }

  // Check if this operator should be redrawn
  // this condition also prevents the property window
  // pop up twice.
  if (isDirty()) {
    return;
  }

  if ( e.getModifiers() == e.BUTTON3_MASK) {
    for (int i=0; i<labels.length; ++i) {
      String fieldText="";

      // Assigned the filed name
      propertyLabels[i] = new JLabel(labels[i]);
      // Get the filed information
      switch (i) {
        case 0:           // name
          fieldText = new String(getName());
          break;
        case 1:           // position X
          fieldText = String.valueOf(getX());
          break;
        case 2:           // position Y
          fieldText = String.valueOf(getY());
          break;
        case 3:           // relation 1
          fieldText = relation1;
          break;
        case 4:           // relation 2
          fieldText = relation2;
          break;
        default:
      }
```

```
        propertyTextFields[i] = new JTextField(new String(fieldText));
      }

      // System.out.println("popup the property window");

      PropertyWindow propertyWindow = new
PropertyWindow(FrameMain.parent,
                                                    propertyLabels,

propertyTextFields);

      if (propertyWindow.propertyOption == propertyWindow.OK) {
        // Save the changes
        for (int i=0; i<labels.length; ++i) {

          String fieldText = propertyTextFields[i].getText();

          // Get the field inforamtion from the text field
          switch (i) {
            case 0:           // name
              setName(fieldText);
              break;
            case 1:           // position X
              setX((new Double(fieldText)).doubleValue());
              break;
            case 2:           // position Y
              setY((new Double(fieldText)).doubleValue());
              break;
            case 3:           // relation 1
              setRelation1(fieldText);
              break;
            case 4:           // relation 2
              setRelation2(fieldText);
              break;
            default:
          }

        }

      }
      propertyWindow.dispose();

      // this operator should be redrawn
      setDirty(true);
    }

  }

  public void draw(Graphics g) {
    if (g == null) {
      System.out.println("graphics is null on OperatorDiff->draw()");
      return;
    }

    // Draw the main body
    super.draw(g);
    // Draw the nodes
    super.drawTwoInputNodes(g);
```

```
    // Draw the relation 1 name
    drawRelation1(g);
    // Draw the relation 2 name
    drawRelation2(g);

    // now, the operator should not be redrawn again
    super.setDirty(false);

  }

  public void drawRelation1(Graphics graphics) {

    double relationX = 0;
    double relationY = 0;
    String relation = "";

    relation = relation1;

    if (graphics == null) {
      return;
    }

    if (relation.length() == 0) {
      return;
    }

    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;

    // Make sure this oper
    if (isInCollection(relation)) {
      // Draw the connection line to output node (ox, oy) on relation
      // System.out.println(relation + " is in the collection");
      Operator op = (Operator) getObject(relation);

      g.drawLine((int) (op.ox + Operator.NODE_RADIUS/2),
                 (int) (op.oy + Operator.NODE_RADIUS),
                 (int) (this.x1 + Operator.NODE_RADIUS/2),
                 (int) (this.y1));

    } else {
      // Draw the operator name and connection line
      //System.out.println(relation + " is not in the collection");

      relationX = getX();
      relationY = getY() - getHeight();


      // draw the node
      Shape relationNode = new Ellipse2D.Double(relationX,
                                                relationY,

Operator.NODE_RADIUS,NODE_RADIUS);

      g.draw(relationNode);

      // Draw the conection between two nodes
```

```
    g.drawLine((int) (x1 + Operator.NODE_RADIUS/2),
               (int) y1,
               (int) (relationX + Operator.NODE_RADIUS/2),
               (int) (relationY + Operator.NODE_RADIUS));

    // Calculate the length of the name

    Font holdFont = graphics.getFont();
    graphics.setFont(new Font(holdFont.getName(),
                              holdFont.getStyle(),
                              holdFont.getSize() - 2));
    FontMetrics fm = graphics.getFontMetrics();

    // System.out.println("Font size = " + holdFont.getSize());

    double nameWidth = fm.stringWidth(relation) + 2;
    double nameHeight = fm.getHeight() + 6;

    // draw a straight line
    g.drawLine((int) (relationX - (nameWidth/2)),(int) relationY,
               (int) (relationX + (nameWidth/2)),(int) relationY);

    // draw the relation name
    g.drawString(relation,
                 (int) (relationX - (nameWidth/2) + 1),
                 (int) (relationY - 3));


   // after all, change the font back to original
   graphics.setFont(holdFont);

  }


}

public void drawRelation2(Graphics graphics) {

  double relationX = 0;
  double relationY = 0;

  String relation = "";

  relation = relation2;

  if (graphics == null) {
    return;
  }

  if (relation.length() == 0) {
    return;
  }

  // Use the Graphics2D object
  Graphics2D g = (Graphics2D) graphics;

  // Make sure this oper
  if (isInCollection(relation)) {
```

```
        // Draw the connection line to output node (ox, oy) on relation
        // System.out.println(relation + " is in the collection");
        Operator op = (Operator) getObject(relation);

        g.drawLine((int) (op.ox + Operator.NODE_RADIUS/2),
                   (int) (op.oy + Operator.NODE_RADIUS),
                   (int) (this.x2 + Operator.NODE_RADIUS/2),
                   (int) (this.y2));

      } else {
        // Draw the operator name and connection line
        //System.out.println(relation + " is not in the collection");

        relationX = getX() + getWidth();
        relationY = getY() - getHeight();


        // draw the node
        Shape relationNode = new Ellipse2D.Double(relationX,
                                                  relationY,

Operator.NODE_RADIUS,NODE_RADIUS);

        g.draw(relationNode);

        // Draw the conection between two nodes
        g.drawLine((int) (x2 + Operator.NODE_RADIUS/2),
                   (int) y2,
                   (int) (relationX + Operator.NODE_RADIUS/2),
                   (int) (relationY + Operator.NODE_RADIUS));

        // Calculate the length of the name

        Font holdFont = graphics.getFont();
        graphics.setFont(new Font(holdFont.getName(),
                                  holdFont.getStyle(),
                                  holdFont.getSize() - 2));
        FontMetrics fm = graphics.getFontMetrics();

        // System.out.println("Font size = " + holdFont.getSize());

        double nameWidth = fm.stringWidth(relation) + 2;
        double nameHeight = fm.getHeight() + 6;

        // draw a straight line
        g.drawLine((int) (relationX - (nameWidth/2)),(int) relationY,
                   (int) (relationX + (nameWidth/2)),(int) relationY);

        // draw the relation name
        g.drawString(relation,
                     (int) (relationX - (nameWidth/2) + 1),
                     (int) (relationY - 3));


       // after all, change the font back to original
       graphics.setFont(holdFont);

      }
```

```
  }

  // Action area
  public String buildQuery() {
    StringBuffer qry = new StringBuffer("select distinct * from ");
    // Check if the relation is operator object in the collection
    if (isInCollection(relation1)) {
      qry.append(" (" );
      Operator op = (Operator) getObject(relation1);
      qry.append(op.buildQuery());
      qry.append(")");
    } else {
      qry.append(relation1);
    }
    qry.append(" minus ");
    qry.append("select distinct * from ");
    if (isInCollection(relation2)) {
      qry.append(" (" );
      Operator op = (Operator) getObject(relation2);
      qry.append(op.buildQuery());
      qry.append(")");
    } else {
      qry.append(relation2);
    }

    return qry.toString();
  }


  // Implement Externalizable interface - write
  public void writeExternal(ObjectOutput out) throws IOException {
    // Call the super class to save the common data
    super.writeExternal(out);

    // Write the data belong to this operator
    out.writeObject(relation1);
    out.writeObject(relation2);

  }


  // Implement Externalizable interface - read
  public void readExternal(ObjectInput in) {
    // call the super class to get the common data
    super.readExternal(in);

    // read the data belongs to this operator
    try {
      relation1 = (String) in.readObject();
      relation2 = (String) in.readObject();
   } catch (Exception e) {
      System.err.println(e);
   }
  }
}
```

13. OperatorGroupALLsatisfy.java

```
/*
 * Author: Ron Chen
 * File: OperatorGroupALLsatisfy.java
 *
 * A Class for groupALLsatisfy operator for DFQL
 *
*/

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import java.io.*;
import java.util.*;


public class OperatorGroupALLsatisfy extends Operator
{

  final static String OPERATOR_TYPE = "groupALLsatisfy";
  String[] labels = {"Name", "Position X", "Position Y",
                     "Relation", "Grouping Attributes", "Condition"};
  JLabel[] propertyLabels = new JLabel[labels.length];
  JTextField[] propertyTextFields = new JTextField[labels.length];

  // ------ Property -----------------
  private String relation="";
  private String groupAttributes="";
  private String condition="";

  // ------ end of property ----------

  public OperatorGroupALLsatisfy() {
    super(OPERATOR_TYPE);
  }

  public OperatorGroupALLsatisfy(Vector vRef) {
    super(OPERATOR_TYPE, vRef);
  }

  // -------- Property get/set ------------
  public String getRelation() {
    return relation;
  }

  public void setRelation(String sRelation) {
    relation = new String(sRelation);
  }

  public String getGroupAttributes() {
    return groupAttributes;
  }

  public void setGroupAttributes(String sGroupAttributes) {
    groupAttributes = new String(sGroupAttributes);
  }
```

```
public String getCondition() {
  return condition;
}

public void setCondition(String sCondition) {
  if (!(sCondition == null)) {
    condition = new String(sCondition);
  }
}
// --------- end of property get/set -------

// This method is specific used by OperatorUsre.java class
// when the input node node link to the InputBar Node
// when the buildQuery() method, the value of each link node
// will pass to the current operator.  Once the query is built,
// the input node value will be reset to the original value
public void setInputNodeValue(int nNode, String value) {
  // empty body - should extended by each child class
  switch (nNode) {
    case 1:          // relation
      setRelation(value);
      break;
    case 2:          // group attributes
      setGroupAttributes(value);
      break;
    case 3:          // condition
      setCondition(value);
      break;
    default:
  }
}

// mouse events on this class
public void mouseClicked(MouseEvent e) {
  // call the base class event
  super.mouseClicked(e);

  // if this is right mouse clicked, then popup
  // property window
  if (!isInBound(e.getX(), e.getY())) {
    return ;
  }

  // Check if this operator should be redrawn
  // this condition also prevents the property window
  // pop up twice.
  if (isDirty()) {
    return;
  }

  if ( e.getModifiers() == e.BUTTON3_MASK) {

    for (int i=0; i<labels.length; ++i) {
      String fieldText="";

      // Assigned the filed name
      propertyLabels[i] = new JLabel(labels[i]);
```

```
        // Get the filed information
        switch (i) {
          case 0:           // name
            fieldText = new String(getName());
            break;
          case 1:           // position X
            fieldText = String.valueOf(getX());
            break;
          case 2:           // position Y
            fieldText = String.valueOf(getY());
            break;
          case 3:           // relation
            fieldText = relation;
            break;
          case 4:           // group attributes
            fieldText = groupAttributes;
            break;
          case 5:           // condition
            fieldText = condition;
            break;
          default:
        }
        propertyTextFields[i] = new JTextField(new String(fieldText));
      }

      // System.out.println("popup the property window");

      PropertyWindow propertyWindow = new
PropertyWindow(FrameMain.parent,
                                                    propertyLabels,
propertyTextFields);

      if (propertyWindow.propertyOption == propertyWindow.OK) {
        // Save the changes
        for (int i=0; i<labels.length; ++i) {

          String fieldText = propertyTextFields[i].getText();

          // Get the field inforamtion from the text field
          switch (i) {
            case 0:           // name
              setName(fieldText);
              break;
            case 1:           // position X
              setX((new Double(fieldText)).doubleValue());
              break;
            case 2:           // position Y
              setY((new Double(fieldText)).doubleValue());
              break;
            case 3:           // relation
              setRelation(fieldText);
              break;
            case 4:           // group attributes
              setGroupAttributes(fieldText);
              break;
            case 5:           // condition
              setCondition(fieldText);
```

```
            break;
          default:
        }

      }

    }
    propertyWindow.dispose();

    // this operator should be redrawn
    setDirty(true);
  }

}

public void draw(Graphics g) {
  if (g == null) {
    return;
  }

  // Draw the main body
  super.draw(g);
  // Draw the nodes
  super.drawThreeInputNodes(g);
  // Draw the relation name
  drawRelation(g);
  // Draw the group attributes name
  drawGroupAttributes(g);

  // Draw the Count Attributes string
  drawCondition(g);

  // now, the operator should not be redrawn again
  super.setDirty(false);

}

public void drawRelation(Graphics graphics) {

  double relationX = 0;
  double relationY = 0;

  if (graphics == null) {
    return;
  }

  if (relation.length() == 0) {
    return;
  }


  // Use the Graphics2D object
  Graphics2D g = (Graphics2D) graphics;

  // Make sure this oper
  if (isInCollection(relation)) {
    // Draw the connection line to output node (ox, oy) on relation
    // System.out.println(relation + " is in the collection");
```

```
        Operator op = (Operator) getObject(relation);

        g.drawLine((int) (op.ox + Operator.NODE_RADIUS/2),
                   (int) (op.oy + Operator.NODE_RADIUS),
                   (int) (this.x1 + Operator.NODE_RADIUS/2),
                   (int) (this.y1));

      } else {
        // Draw the operator name and connection line
        //System.out.println(relation + " is not in the collection");

        relationX = x1 - getWidth()/2;
        relationY = getY() - getHeight();


        // draw the node
        Shape relationNode = new Ellipse2D.Double(relationX,
                                                  relationY,

Operator.NODE_RADIUS,NODE_RADIUS);

        g.draw(relationNode);

        // Draw the conection between two nodes
        g.drawLine((int) (x1 + Operator.NODE_RADIUS/2),
                   (int) y1,
                   (int) (relationX + Operator.NODE_RADIUS/2),
                   (int) (relationY + Operator.NODE_RADIUS));

        // Calculate the length of the name

        Font holdFont = graphics.getFont();
        graphics.setFont(new Font(holdFont.getName(),
                                  holdFont.getStyle(),
                                  holdFont.getSize() - 2));
        FontMetrics fm = graphics.getFontMetrics();

        // System.out.println("Font size = " + holdFont.getSize());

        double nameWidth = fm.stringWidth(relation) + 2;
        double nameHeight = fm.getHeight() + 6;

        // draw a straight line
        g.drawLine((int) (relationX - (nameWidth/2)),(int) relationY,
                   (int) (relationX + (nameWidth/2)),(int) relationY);

        // draw the relation name
        g.drawString(relation,
                     (int) (relationX - (nameWidth/2) + 1),
                     (int) (relationY - 3));


       // after all, change the font back to original
       graphics.setFont(holdFont);

      }
```

```
  }

  public void drawGroupAttributes(Graphics graphics) {

    double groupX = 0;
    double groupY = 0;


    if (graphics == null) {
      return;
    }

    if (groupAttributes.length() == 0) {
      return;
    }


    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;


    groupX = x2;
    groupY = y2 - getHeight() * 2;   // twice's high


    // draw the node
    Shape groupNode = new Ellipse2D.Double(groupX,
                                           groupY,
Operator.NODE_RADIUS,NODE_RADIUS);

    g.draw(groupNode);

    // Draw the conection between two nodes
    g.drawLine((int) (x2 + Operator.NODE_RADIUS/2),
               (int) y2,
               (int) (groupX + Operator.NODE_RADIUS/2),
               (int) (groupY + Operator.NODE_RADIUS));

    // Calculate the length of the name

    Font holdFont = graphics.getFont();
    graphics.setFont(new Font(holdFont.getName(),
                              holdFont.getStyle(),
                              holdFont.getSize() - 2));
    FontMetrics fm = graphics.getFontMetrics();

    // System.out.println("Font size = " + holdFont.getSize());

    double nameWidth = fm.stringWidth(groupAttributes) + 2;
    double nameHeight = fm.getHeight() + 6;

    // draw a straight line
    g.drawLine((int) (groupX - (nameWidth/2)),(int) groupY,
               (int) (groupX + (nameWidth/2)),(int) groupY);

    // draw the relation name
    g.drawString(groupAttributes,
```

```
                (int) (groupX - (nameWidth/2) + 1),
                (int) (groupY - 3));


    // after all, change the font back to original
    graphics.setFont(holdFont);

  }


  public void drawCondition(Graphics graphics) {

    double conditionX = 0;
    double conditionY = 0;

    if (graphics == null) {
      return;
    }

    if (condition.length() == 0) {
      return;
    }

    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;

    // Draw the condition name and connection line
    conditionX = x3 + getWidth()/2;
    conditionY = getY() - getHeight();


    // draw the node
    Shape conditionNode = new Ellipse2D.Double(conditionX,
                                               conditionY,

Operator.NODE_RADIUS,NODE_RADIUS);

    g.draw(conditionNode);

    // Draw the conection between two nodes
    g.drawLine((int) (x3 + Operator.NODE_RADIUS/2),
               (int) y3,
               (int) (conditionX + Operator.NODE_RADIUS/2),
               (int) (conditionY + Operator.NODE_RADIUS));

    // Calculate the length of the name

    Font holdFont = graphics.getFont();
    graphics.setFont(new Font(holdFont.getName(),
                              holdFont.getStyle(),
                              holdFont.getSize() - 2));
    FontMetrics fm = graphics.getFontMetrics();

    // System.out.println("Font size = " + holdFont.getSize());

    double nameWidth = fm.stringWidth(condition) + 2;
    double nameHeight = fm.getHeight() + 6;
```

```
    // draw a straight line
    g.drawLine((int) (conditionX - (nameWidth/2)),(int) conditionY,
               (int) (conditionX + (nameWidth/2)),(int) conditionY);

    // draw the relation name
    g.drawString(condition,
                 (int) (conditionX - (nameWidth/2) + 1),
                 (int) (conditionY - 3));


   // after all, change the font back to original
   graphics.setFont(holdFont);

  }

  // Action area
  public String buildQuery() {
    StringBuffer qry = new StringBuffer("select distinct ");
    if (groupAttributes.length() != 0) {
      qry.append(groupAttributes);
    }
    else {
      return (new String(""));
    }

    qry.append(" from ");
    // Check if the relation is operator object in the collection
    if (isInCollection(relation)) {
      qry.append(" (" );
      Operator op = (Operator) getObject(relation);
      qry.append(op.buildQuery());
      qry.append(")");
    } else {
      qry.append(relation);
    }
    // check condition
    if (condition.length() > 0) {
      qry.append(" where ").append(condition);

    }
    // group by
    if (groupAttributes.length() > 0) {
      qry.append(" group by ").append(groupAttributes);
    }

    return qry.toString();
  }


  // Implement Externalizable interface - write
  public void writeExternal(ObjectOutput out) throws IOException {
    // Call the super class to save the common data
    super.writeExternal(out);

    // Write the data belong to this operator
    out.writeObject(relation);
    out.writeObject(groupAttributes);
    out.writeObject(condition);
```

```
  }

  // Implement Externalizable interface - read
  public void readExternal(ObjectInput in) {
    // call the super class to get the common data
    super.readExternal(in);

    // read the data belongs to this operator
    try {
      relation = (String) in.readObject();
      groupAttributes = (String) in.readObject();
      condition = (String) in.readObject();
   } catch (Exception e) {
      System.err.println(e);
   }
  }
}
```

14. OperatorGroupavg.java

```
/*
 * Author: Ron Chen
 * File: OperatorGroupavg.java
 *
 * A Class for groupavg operator for DFQL
 *
*/

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import java.io.*;
import java.util.*;


public class OperatorGroupavg extends Operator
{

  final static String OPERATOR_TYPE = "groupavg";
  String[] labels = {"Name", "Position X", "Position Y",
                     "Relation", "Grouping Attributes", "Aggregate
Attribute"};
  JLabel[] propertyLabels = new JLabel[labels.length];
  JTextField[] propertyTextFields = new JTextField[labels.length];

  // ------ Property -----------------
  private String relation="";
  private String groupAttributes="";
  private String aggregateAttribute="";

  // ------ end of property ----------
  public OperatorGroupavg() {
```

```
  super(OPERATOR_TYPE);
}

public OperatorGroupavg(Vector vRef) {
  super(OPERATOR_TYPE, vRef);
}

// -------- Property get/set ------------
public String getRelation() {
  return relation;
}

public void setRelation(String sRelation) {
  relation = new String(sRelation);
}

public String getGroupAttributes() {
  return groupAttributes;
}

public void setGroupAttributes(String sGroupAttributes) {
  groupAttributes = new String(sGroupAttributes);
}

public String getAggregateAttribute() {
  return aggregateAttribute;
}

public void setAggregateAttribute(String sAggregateAttribute) {
  if (!(sAggregateAttribute == null)) {
    aggregateAttribute = new String(sAggregateAttribute);
  }
}
// --------- end of property get/set -------

// This method is specific used by OperatorUsre.java class
// when the input node node link to the InputBar Node
// when the buildQuery() method, the value of each link node
// will pass to the current operator.  Once the query is built,
// the input node value will be reset to the original value
public void setInputNodeValue(int nNode, String value) {
  // empty body - should extended by each child class
  switch (nNode) {
    case 1:          // relation
      setRelation(value);
      break;
    case 2:          // group attributes
      setGroupAttributes(value);
      break;
    case 3:          // aggregate attribute
      setAggregateAttribute(value);
      break;
    default:
  }
}

// mouse events on this class
public void mouseClicked(MouseEvent e) {
```

```
    // call the base class event
    super.mouseClicked(e);

    // if this is right mouse clicked, then popup
    // property window
    if (!isInBound(e.getX(), e.getY())) {
      return ;
    }

    // Check if this operator should be redrawn
    // this condition also prevents the property window
    // pop up twice.
    if (isDirty()) {
      return;
    }

    if ( e.getModifiers() == e.BUTTON3_MASK) {

      for (int i=0; i<labels.length; ++i) {
        String fieldText="";

        // Assigned the filed name
        propertyLabels[i] = new JLabel(labels[i]);
        // Get the filed information
        switch (i) {
          case 0:           // name
            fieldText = new String(getName());
            break;
          case 1:           // position X
            fieldText = String.valueOf(getX());
            break;
          case 2:           // position Y
            fieldText = String.valueOf(getY());
            break;
          case 3:           // relation
            fieldText = relation;
            break;
          case 4:           // group attributes
            fieldText = groupAttributes;
            break;
          case 5:           // aggregate attribute
            fieldText = aggregateAttribute;
            break;
          default:
        }
        propertyTextFields[i] = new JTextField(new String(fieldText));
      }

      // System.out.println("popup the property window");

      PropertyWindow propertyWindow = new
PropertyWindow(FrameMain.parent,
                                                    propertyLabels,

propertyTextFields);

      if (propertyWindow.propertyOption == propertyWindow.OK) {
        // Save the changes
```

```
      for (int i=0; i<labels.length; ++i) {

        String fieldText = propertyTextFields[i].getText();

        // Get the field inforamtion from the text field
        switch (i) {
          case 0:           // name
            setName(fieldText);
            break;
          case 1:           // position X
            setX((new Double(fieldText)).doubleValue());
            break;
          case 2:           // position Y
            setY((new Double(fieldText)).doubleValue());
            break;
          case 3:           // relation
            setRelation(fieldText);
            break;
          case 4:           // group attributes
            setGroupAttributes(fieldText);
            break;
          case 5:           // aggregate attributes
            setAggregateAttribute(fieldText);
            break;
          default:
        }

      }

    }
    propertyWindow.dispose();

    // this operator should be redrawn
    setDirty(true);
  }

}
public void draw(Graphics g) {
  if (g == null) {
    return;
  }

  // Draw the main body
  super.draw(g);
  // Draw the nodes
  super.drawThreeInputNodes(g);
  // Draw the relation name
  drawRelation(g);
  // Draw the group attributes name
  drawGroupAttributes(g);

  // Draw the Aggregate Attributes string
  drawAggregate(g);

  // now, the operator should not be redrawn again
  super.setDirty(false);

}
```

```
  public void drawRelation(Graphics graphics) {

    double relationX = 0;
    double relationY = 0;

    if (graphics == null) {
      return;
    }

    if (relation.length() == 0) {
      return;
    }


    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;

    // Make sure this oper
    if (isInCollection(relation)) {
      // Draw the connection line to output node (ox, oy) on relation
      // System.out.println(relation + " is in the collection");
      Operator op = (Operator) getObject(relation);

      g.drawLine((int) (op.ox + Operator.NODE_RADIUS/2),
                 (int) (op.oy + Operator.NODE_RADIUS),
                 (int) (this.x1 + Operator.NODE_RADIUS/2),
                 (int) (this.y1));

    } else {
      // Draw the operator name and connection line
      //System.out.println(relation + " is not in the collection");

      relationX = x1 - getWidth()/2;
      relationY = getY() - getHeight();


      // draw the node
      Shape relationNode = new Ellipse2D.Double(relationX,
                                                relationY,

Operator.NODE_RADIUS,NODE_RADIUS);

      g.draw(relationNode);

      // Draw the conection between two nodes
      g.drawLine((int) (x1 + Operator.NODE_RADIUS/2),
                 (int) y1,
                 (int) (relationX + Operator.NODE_RADIUS/2),
                 (int) (relationY + Operator.NODE_RADIUS));

      // Calculate the length of the name

      Font holdFont = graphics.getFont();
      graphics.setFont(new Font(holdFont.getName(),
                                holdFont.getStyle(),
                                holdFont.getSize() - 2));
      FontMetrics fm = graphics.getFontMetrics();
```

```
        // System.out.println("Font size = " + holdFont.getSize());

        double nameWidth = fm.stringWidth(relation) + 2;
        double nameHeight = fm.getHeight() + 6;

        // draw a straight line
        g.drawLine((int) (relationX - (nameWidth/2)),(int) relationY,
                   (int) (relationX + (nameWidth/2)),(int) relationY);

        // draw the relation name
        g.drawString(relation,
                     (int) (relationX - (nameWidth/2) + 1),
                     (int) (relationY - 3));


       // after all, change the font back to original
       graphics.setFont(holdFont);

      }


    }

    public void drawGroupAttributes(Graphics graphics) {

      double groupX = 0;
      double groupY = 0;


      if (graphics == null) {
        return;
      }

      if (groupAttributes.length() == 0) {
        return;
      }


      // Use the Graphics2D object
      Graphics2D g = (Graphics2D) graphics;


      groupX = x2;
      groupY = y2 - getHeight() * 2;   // twice's high


      // draw the node
      Shape groupNode = new Ellipse2D.Double(groupX,
                                             groupY,
Operator.NODE_RADIUS,NODE_RADIUS);

      g.draw(groupNode);

      // Draw the conection between two nodes
      g.drawLine((int) (x2 + Operator.NODE_RADIUS/2),
                 (int) y2,
```

```
                (int) (groupX + Operator.NODE_RADIUS/2),
                (int) (groupY + Operator.NODE_RADIUS));

    // Calculate the length of the name

    Font holdFont = graphics.getFont();
    graphics.setFont(new Font(holdFont.getName(),
                              holdFont.getStyle(),
                              holdFont.getSize() - 2));
    FontMetrics fm = graphics.getFontMetrics();

    // System.out.println("Font size = " + holdFont.getSize());

    double nameWidth = fm.stringWidth(groupAttributes) + 2;
    double nameHeight = fm.getHeight() + 6;

    // draw a straight line
    g.drawLine((int) (groupX - (nameWidth/2)),(int) groupY,
               (int) (groupX + (nameWidth/2)),(int) groupY);

    // draw the relation name
    g.drawString(groupAttributes,
                 (int) (groupX - (nameWidth/2) + 1),
                 (int) (groupY - 3));


   // after all, change the font back to original
   graphics.setFont(holdFont);

  }


  public void drawAggregate(Graphics graphics) {

    double aggregateX = 0;
    double aggregateY = 0;

    if (graphics == null) {
      return;
    }

    if (aggregateAttribute.length() == 0) {
      return;
    }

    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;

    // Draw the condition name and connection line
    aggregateX = x3 + getWidth()/2;
    aggregateY = getY() - getHeight();


    // draw the node
    Shape aggregateNode = new Ellipse2D.Double(aggregateX,
                                               aggregateY,

Operator.NODE_RADIUS,NODE_RADIUS);
```

```
  g.draw(aggregateNode);

  // Draw the conection between two nodes
  g.drawLine((int) (x3 + Operator.NODE_RADIUS/2),
             (int) y3,
             (int) (aggregateX + Operator.NODE_RADIUS/2),
             (int) (aggregateY + Operator.NODE_RADIUS));

  // Calculate the length of the name

  Font holdFont = graphics.getFont();
  graphics.setFont(new Font(holdFont.getName(),
                            holdFont.getStyle(),
                            holdFont.getSize() - 2));
  FontMetrics fm = graphics.getFontMetrics();

  // System.out.println("Font size = " + holdFont.getSize());

  double nameWidth = fm.stringWidth(aggregateAttribute) + 2;
  double nameHeight = fm.getHeight() + 6;

  // draw a straight line
  g.drawLine((int) (aggregateX - (nameWidth/2)),(int) aggregateY,
             (int) (aggregateX + (nameWidth/2)),(int) aggregateY);

  // draw the aggregate name
  g.drawString(aggregateAttribute,
               (int) (aggregateX - (nameWidth/2) + 1),
               (int) (aggregateY - 3));


 // after all, change the font back to original
 graphics.setFont(holdFont);

}

// Action area
public String buildQuery() {
  StringBuffer qry = new StringBuffer("select distinct ");
  if (groupAttributes.length() != 0) {
    qry.append(groupAttributes);
  }
  if (aggregateAttribute.length() != 0) {
    if (groupAttributes.length() != 0) {
      qry.append(", ");
    }
    // MAX keywork of the aggregate attributes
    qry.append("avg(").append(aggregateAttribute).append(")");
  } else {
    return (new String(""));
  }

  qry.append(" from ");
  // Check if the relation is operator object in the collection
  if (isInCollection(relation)) {
    qry.append(" (" );
    Operator op = (Operator) getObject(relation);
```

```
      qry.append(op.buildQuery());
      qry.append(")");
    } else {
      qry.append(relation);
    }
    // group by
    if (groupAttributes.length() > 0) {
      qry.append(" group by ").append(groupAttributes);
    }

    return qry.toString();
  }


  // Implement Externalizable interface - write
  public void writeExternal(ObjectOutput out) throws IOException {
    // Call the super class to save the common data
    super.writeExternal(out);

    // Write the data belong to this operator
    out.writeObject(relation);
    out.writeObject(groupAttributes);
    out.writeObject(aggregateAttribute);

  }


  // Implement Externalizable interface - read
  public void readExternal(ObjectInput in) {
    // call the super class to get the common data
    super.readExternal(in);

    // read the data belongs to this operator
    try {
      relation = (String) in.readObject();
      groupAttributes = (String) in.readObject();
      aggregateAttribute = (String) in.readObject();
   } catch (Exception e) {
      System.err.println(e);
    }
  }
}
```

15. OperatorGroupcnt.java

```
/*
 * Author: Ron Chen
 * File: OperatorGroupcnt.java
 *
 * A Class for groupcnt operator for DFQL
 *
*/

import java.awt.*;
import java.awt.event.*;
```

```
import java.awt.geom.*;
import javax.swing.*;
import java.io.*;
import java.util.*;


public class OperatorGroupcnt extends Operator
{

  final static String OPERATOR_TYPE = "groupcnt";
  String[] labels = {"Name", "Position X", "Position Y",
                     "Relation", "Grouping Attributes", "Count
Attributes"};
  JLabel[] propertyLabels = new JLabel[labels.length];
  JTextField[] propertyTextFields = new JTextField[labels.length];

  // ------ Property -----------------
  private String relation="";
  private String groupAttributes="";
  private String countAttribute="";

  // ------ end of property ----------

  public OperatorGroupcnt() {
    super(OPERATOR_TYPE);
  }

  public OperatorGroupcnt(Vector vRef) {
    super(OPERATOR_TYPE, vRef);
  }

  // -------- Property get/set ------------
  public String getRelation() {
    return relation;
  }

  public void setRelation(String sRelation) {
    relation = new String(sRelation);
  }

  public String getGroupAttributes() {
    return groupAttributes;
  }

  public void setGroupAttributes(String sGroupAttributes) {
    groupAttributes = new String(sGroupAttributes);
  }

  public String getCountAttributes() {
    return countAttribute;
  }

  public void setCountAttributes(String sCountAttributes) {
    if (!(sCountAttributes == null)) {
      countAttribute = new String(sCountAttributes);
    }
  }
  // --------- end of property get/set -------
```

```
// This method is specific used by OperatorUsre.java class
// when the input node node link to the InputBar Node
// when the buildQuery() method, the value of each link node
// will pass to the current operator.  Once the query is built,
// the input node value will be reset to the original value
public void setInputNodeValue(int nNode, String value) {
  // empty body - should extended by each child class
  switch (nNode) {
    case 1:         // relation
      setRelation(value);
      break;
    case 2:         // group attributes
      setGroupAttributes(value);
      break;
    case 3:         // count attributes
      setCountAttributes(value);
      break;
    default:
  }
}

// mouse events on this class
public void mouseClicked(MouseEvent e) {
  // call the base class event
  super.mouseClicked(e);

  // if this is right mouse clicked, then popup
  // property window
  if (!isInBound(e.getX(), e.getY())) {
    return ;
  }

  // Check if this operator should be redrawn
  // this condition also prevents the property window
  // pop up twice.
  if (isDirty()) {
    return;
  }

  if ( e.getModifiers() == e.BUTTON3_MASK) {

    for (int i=0; i<labels.length; ++i) {
      String fieldText="";

      // Assigned the filed name
      propertyLabels[i] = new JLabel(labels[i]);
      // Get the filed information
      switch (i) {
        case 0:           // name
          fieldText = new String(getName());
          break;
        case 1:           // position X
          fieldText = String.valueOf(getX());
          break;
        case 2:           // position Y
          fieldText = String.valueOf(getY());
          break;
```

```
            case 3:             // relation
              fieldText = relation;
              break;
            case 4:             // group attributes
              fieldText = groupAttributes;
              break;
            case 5:             // count attributes
              fieldText = countAttribute;
              break;
            default:
          }
          propertyTextFields[i] = new JTextField(new String(fieldText));
        }

        // System.out.println("popup the property window");

        PropertyWindow propertyWindow = new
PropertyWindow(FrameMain.parent,
                                                        propertyLabels,

propertyTextFields);

        if (propertyWindow.propertyOption == propertyWindow.OK) {
          // Save the changes
          for (int i=0; i<labels.length; ++i) {

            String fieldText = propertyTextFields[i].getText();

            // Get the field inforamtion from the text field
            switch (i) {
              case 0:             // name
                setName(fieldText);
                break;
              case 1:             // position X
                setX((new Double(fieldText)).doubleValue());
                break;
              case 2:             // position Y
                setY((new Double(fieldText)).doubleValue());
                break;
              case 3:             // relation
                setRelation(fieldText);
                break;
              case 4:             // group attributes
                setGroupAttributes(fieldText);
                break;
              case 5:             // count attributes
                setCountAttributes(fieldText);
                break;
              default:
            }

          }

        }
        propertyWindow.dispose();

        // this operator should be redrawn
        setDirty(true);
```

```
    }

  }


  public void draw(Graphics g) {
    if (g == null) {
      System.out.println("graphics is null on OperatorGroupcnt-
>draw()");
      return;
    }

    // Draw the main body
    super.draw(g);
    // Draw the nodes
    super.drawThreeInputNodes(g);

    // Draw the relation name
    drawRelation(g);
    // Draw the group attributes name
    drawGroupAttributes(g);

    // Draw the Count Attributes string
    drawCountAttributes(g);

    // now, the operator should not be redrawn again
    super.setDirty(false);

  }

  public void drawRelation(Graphics graphics) {

    double relationX = 0;
    double relationY = 0;

    if (graphics == null) {
      return;
    }

    if (relation.length() == 0) {
      return;
    }


    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;

    // Make sure this oper
    if (isInCollection(relation)) {
      // Draw the connection line to output node (ox, oy) on relation
      // System.out.println(relation + " is in the collection");
      Operator op = (Operator) getObject(relation);

      g.drawLine((int) (op.ox + Operator.NODE_RADIUS/2),
                 (int) (op.oy + Operator.NODE_RADIUS),
                 (int) (this.x1 + Operator.NODE_RADIUS/2),
                 (int) (this.y1));
```

```
    } else {
      // Draw the operator name and connection line
      //System.out.println(relation + " is not in the collection");

      relationX = x1 - getWidth()/2;
      relationY = getY() - getHeight();


      // draw the node
      Shape relationNode = new Ellipse2D.Double(relationX,
                                                relationY,
Operator.NODE_RADIUS,NODE_RADIUS);

      g.draw(relationNode);

      // Draw the conection between two nodes
      g.drawLine((int) (x1 + Operator.NODE_RADIUS/2),
                 (int) y1,
                 (int) (relationX + Operator.NODE_RADIUS/2),
                 (int) (relationY + Operator.NODE_RADIUS));

      // Calculate the length of the name

      Font holdFont = graphics.getFont();
      graphics.setFont(new Font(holdFont.getName(),
                                holdFont.getStyle(),
                                holdFont.getSize() - 2));
      FontMetrics fm = graphics.getFontMetrics();

      // System.out.println("Font size = " + holdFont.getSize());

      double nameWidth = fm.stringWidth(relation) + 2;
      double nameHeight = fm.getHeight() + 6;

      // draw a straight line
      g.drawLine((int) (relationX - (nameWidth/2)),(int) relationY,
                 (int) (relationX + (nameWidth/2)),(int) relationY);

      // draw the relation name
      g.drawString(relation,
                   (int) (relationX - (nameWidth/2) + 1),
                   (int) (relationY - 3));


     // after all, change the font back to original
     graphics.setFont(holdFont);

    }


  }

  public void drawGroupAttributes(Graphics graphics) {

    double groupX = 0;
    double groupY = 0;
```

```
    if (graphics == null) {
      return;
    }

    if (groupAttributes.length() == 0) {
      return;
    }


    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;


    groupX = x2;
    groupY = y2 - getHeight() * 2;   // twice's high


    // draw the node
    Shape groupNode = new Ellipse2D.Double(groupX,
                                           groupY,

Operator.NODE_RADIUS,NODE_RADIUS);

    g.draw(groupNode);

    // Draw the conection between two nodes
    g.drawLine((int) (x2 + Operator.NODE_RADIUS/2),
               (int) y2,
               (int) (groupX + Operator.NODE_RADIUS/2),
               (int) (groupY + Operator.NODE_RADIUS));

    // Calculate the length of the name

    Font holdFont = graphics.getFont();
    graphics.setFont(new Font(holdFont.getName(),
                              holdFont.getStyle(),
                              holdFont.getSize() - 2));
    FontMetrics fm = graphics.getFontMetrics();

    // System.out.println("Font size = " + holdFont.getSize());

    double nameWidth = fm.stringWidth(groupAttributes) + 2;
    double nameHeight = fm.getHeight() + 6;

    // draw a straight line
    g.drawLine((int) (groupX - (nameWidth/2)),(int) groupY,
               (int) (groupX + (nameWidth/2)),(int) groupY);

    // draw the relation name
    g.drawString(groupAttributes,
                 (int) (groupX - (nameWidth/2) + 1),
                 (int) (groupY - 3));


  // after all, change the font back to original
  graphics.setFont(holdFont);
```

```
  }

  public void drawCountAttributes(Graphics graphics) {

    double countX = 0;
    double countY = 0;

    if (graphics == null) {
      return;
    }

    if (countAttribute.length() == 0) {
      return;
    }

    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;

    // Draw the condition name and connection line
    countX = x3 + getWidth()/2;
    countY = getY() - getHeight();


    // draw the node
    Shape countNode = new Ellipse2D.Double(countX,
                                           countY,
Operator.NODE_RADIUS,NODE_RADIUS);

    g.draw(countNode);

    // Draw the conection between two nodes
    g.drawLine((int) (x3 + Operator.NODE_RADIUS/2),
               (int) y3,
               (int) (countX + Operator.NODE_RADIUS/2),
               (int) (countY + Operator.NODE_RADIUS));

    // Calculate the length of the name

    Font holdFont = graphics.getFont();
    graphics.setFont(new Font(holdFont.getName(),
                              holdFont.getStyle(),
                              holdFont.getSize() - 2));
    FontMetrics fm = graphics.getFontMetrics();

    // System.out.println("Font size = " + holdFont.getSize());

    double nameWidth = fm.stringWidth(countAttribute) + 2;
    double nameHeight = fm.getHeight() + 6;

    // draw a straight line
    g.drawLine((int) (countX - (nameWidth/2)),(int) countY,
               (int) (countX + (nameWidth/2)),(int) countY);

    // draw the relation name
    g.drawString(countAttribute,
                 (int) (countX - (nameWidth/2) + 1),
```

```
                (int) (countY - 3));

 // after all, change the font back to original
 graphics.setFont(holdFont);

}

// Action area
public String buildQuery() {
  StringBuffer qry = new StringBuffer("select distinct ");
  if (groupAttributes.length() != 0) {
    qry.append(groupAttributes);
    qry.append(", ");
  }
  qry.append("count(*)");
  if (countAttribute.length() > 0) {
    qry.append(" ").append(countAttribute);
  }

  qry.append(" from ");
  // Check if the relation is operator object in the collection
  if (isInCollection(relation)) {
    qry.append(" (" );
    Operator op = (Operator) getObject(relation);
    qry.append(op.buildQuery());
    qry.append(")");
  } else {
    qry.append(relation);
  }

  if (groupAttributes.length() > 0) {
    qry.append(" group by ").append(groupAttributes);
  }

  return qry.toString();
}

// Implement Externalizable interface - write
public void writeExternal(ObjectOutput out) throws IOException {
  // Call the super class to save the common data
  super.writeExternal(out);

  // Write the data belong to this operator
  out.writeObject(relation);
  out.writeObject(groupAttributes);
  out.writeObject(countAttribute);

}


// Implement Externalizable interface - read
public void readExternal(ObjectInput in) {
  // call the super class to get the common data
  super.readExternal(in);

  // read the data belongs to this operator
  try {
```

```
      relation = (String) in.readObject();
      groupAttributes = (String) in.readObject();
      countAttribute = (String) in.readObject();
    } catch (Exception e) {
      System.err.println(e);
    }
  }

}
```

16. OperatorGroupmax.java

```
/*
 * Author: Ron Chen
 * File: OperatorGroupmax.java
 *
 * A Class for groupmax operator for DFQL
 *
*/

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import java.io.*;
import java.util.*;


public class OperatorGroupmax extends Operator
{

  final static String OPERATOR_TYPE = "groupmax";
  String[] labels = {"Name", "Position X", "Position Y",
                     "Relation", "Grouping Attributes", "Aggregate
Attribute"};
  JLabel[] propertyLabels = new JLabel[labels.length];
  JTextField[] propertyTextFields = new JTextField[labels.length];

  // ------ Property -----------------
  private String relation="";
  private String groupAttributes="";
  private String aggregateAttribute="";

  // ------ end of property ----------
  public OperatorGroupmax() {
    super(OPERATOR_TYPE);
  }

  public OperatorGroupmax(Vector vRef) {
    super(OPERATOR_TYPE, vRef);
  }

  // -------- Property get/set ------------
  public String getRelation() {
    return relation;
```

```
}

public void setRelation(String sRelation) {
  relation = new String(sRelation);
}

public String getGroupAttributes() {
  return groupAttributes;
}

public void setGroupAttributes(String sGroupAttributes) {
  groupAttributes = new String(sGroupAttributes);
}

public String getAggregateAttribute() {
  return aggregateAttribute;
}

public void setAggregateAttribute(String sAggregateAttribute) {
  if (!(sAggregateAttribute == null)) {
    aggregateAttribute = new String(sAggregateAttribute);
  }
}
// --------- end of property get/set -------

// This method is specific used by OperatorUsre.java class
// when the input node node link to the InputBar Node
// when the buildQuery() method, the value of each link node
// will pass to the current operator.  Once the query is built,
// the input node value will be reset to the original value
public void setInputNodeValue(int nNode, String value) {
  // empty body - should extended by each child class
  switch (nNode) {
    case 1:          // relation
      setRelation(value);
      break;
    case 2:          // group attributes
      setGroupAttributes(value);
      break;
    case 3:          // aggregate attribute
      setAggregateAttribute(value);
      break;
    default:
  }
}

// mouse events on this class
public void mouseClicked(MouseEvent e) {
  // call the base class event
  super.mouseClicked(e);

  // if this is right mouse clicked, then popup
  // property window
  if (!isInBound(e.getX(), e.getY())) {
    return ;
  }

  // Check if this operator should be redrawn
```

```
    // this condition also prevents the property window
    // pop up twice.
    if (isDirty()) {
      return;
    }

    if ( e.getModifiers() == e.BUTTON3_MASK) {

      for (int i=0; i<labels.length; ++i) {
        String fieldText="";

        // Assigned the filed name
        propertyLabels[i] = new JLabel(labels[i]);
        // Get the filed information
        switch (i) {
          case 0:           // name
            fieldText = new String(getName());
            break;
          case 1:           // position X
            fieldText = String.valueOf(getX());
            break;
          case 2:           // position Y
            fieldText = String.valueOf(getY());
            break;
          case 3:           // relation
            fieldText = relation;
            break;
          case 4:           // group attributes
            fieldText = groupAttributes;
            break;
          case 5:           // aggregate attribute
            fieldText = aggregateAttribute;
            break;
          default:
        }
        propertyTextFields[i] = new JTextField(new String(fieldText));
      }

      // System.out.println("popup the property window");

      PropertyWindow propertyWindow = new
PropertyWindow(FrameMain.parent,
                                                    propertyLabels,
propertyTextFields);

      if (propertyWindow.propertyOption == propertyWindow.OK) {
        // Save the changes
        for (int i=0; i<labels.length; ++i) {

          String fieldText = propertyTextFields[i].getText();

          // Get the field inforamtion from the text field
          switch (i) {
            case 0:           // name
              setName(fieldText);
              break;
            case 1:           // position X
```

```
            setX((new Double(fieldText)).doubleValue());
            break;
          case 2:           // position Y
            setY((new Double(fieldText)).doubleValue());
            break;
          case 3:           // relation
            setRelation(fieldText);
            break;
          case 4:           // group attributes
            setGroupAttributes(fieldText);
            break;
          case 5:           // aggregate attributes
            setAggregateAttribute(fieldText);
            break;
          default:
        }

      }

    }
    propertyWindow.dispose();

    // this operator should be redrawn
    setDirty(true);
  }

}
public void draw(Graphics g) {
  if (g == null) {
    return;
  }

  // Draw the main body
  super.draw(g);
  // Draw the nodes
  super.drawThreeInputNodes(g);
  // Draw the relation name
  drawRelation(g);
  // Draw the group attributes name
  drawGroupAttributes(g);

  // Draw the Aggregate Attributes string
  drawAggregate(g);

  // now, the operator should not be redrawn again
  super.setDirty(false);

}

public void drawRelation(Graphics graphics) {

  double relationX = 0;
  double relationY = 0;

  if (graphics == null) {
    return;
  }
```

```
    if (relation.length() == 0) {
      return;
    }


    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;

    // Make sure this oper
    if (isInCollection(relation)) {
      // Draw the connection line to output node (ox, oy) on relation
      // System.out.println(relation + " is in the collection");
      Operator op = (Operator) getObject(relation);

      g.drawLine((int) (op.ox + Operator.NODE_RADIUS/2),
                 (int) (op.oy + Operator.NODE_RADIUS),
                 (int) (this.x1 + Operator.NODE_RADIUS/2),
                 (int) (this.y1));

    } else {
      // Draw the operator name and connection line
      //System.out.println(relation + " is not in the collection");

      relationX = x1 - getWidth()/2;
      relationY = getY() - getHeight();


      // draw the node
      Shape relationNode = new Ellipse2D.Double(relationX,
                                                relationY,

Operator.NODE_RADIUS,NODE_RADIUS);

      g.draw(relationNode);

      // Draw the conection between two nodes
      g.drawLine((int) (x1 + Operator.NODE_RADIUS/2),
                 (int) y1,
                 (int) (relationX + Operator.NODE_RADIUS/2),
                 (int) (relationY + Operator.NODE_RADIUS));

      // Calculate the length of the name

      Font holdFont = graphics.getFont();
      graphics.setFont(new Font(holdFont.getName(),
                                holdFont.getStyle(),
                                holdFont.getSize() - 2));
      FontMetrics fm = graphics.getFontMetrics();

      // System.out.println("Font size = " + holdFont.getSize());

      double nameWidth = fm.stringWidth(relation) + 2;
      double nameHeight = fm.getHeight() + 6;

      // draw a straight line
      g.drawLine((int) (relationX - (nameWidth/2)),(int) relationY,
                 (int) (relationX + (nameWidth/2)),(int) relationY);
```

```
      // draw the relation name
      g.drawString(relation,
                   (int) (relationX - (nameWidth/2) + 1),
                   (int) (relationY - 3));


     // after all, change the font back to original
     graphics.setFont(holdFont);

    }


  }

  public void drawGroupAttributes(Graphics graphics) {

    double groupX = 0;
    double groupY = 0;


    if (graphics == null) {
      return;
    }

    if (groupAttributes.length() == 0) {
      return;
    }


    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;


    groupX = x2;
    groupY = y2 - getHeight() * 2;   // twice's high


    // draw the node
    Shape groupNode = new Ellipse2D.Double(groupX,
                                           groupY,

Operator.NODE_RADIUS,NODE_RADIUS);

    g.draw(groupNode);

    // Draw the conection between two nodes
    g.drawLine((int) (x2 + Operator.NODE_RADIUS/2),
               (int) y2,
               (int) (groupX + Operator.NODE_RADIUS/2),
               (int) (groupY + Operator.NODE_RADIUS));

    // Calculate the length of the name

    Font holdFont = graphics.getFont();
    graphics.setFont(new Font(holdFont.getName(),
                              holdFont.getStyle(),
                              holdFont.getSize() - 2));
    FontMetrics fm = graphics.getFontMetrics();
```

```
    // System.out.println("Font size = " + holdFont.getSize());

    double nameWidth = fm.stringWidth(groupAttributes) + 2;
    double nameHeight = fm.getHeight() + 6;

    // draw a straight line
    g.drawLine((int) (groupX - (nameWidth/2)),(int) groupY,
               (int) (groupX + (nameWidth/2)),(int) groupY);

    // draw the relation name
    g.drawString(groupAttributes,
                 (int) (groupX - (nameWidth/2) + 1),
                 (int) (groupY - 3));


   // after all, change the font back to original
   graphics.setFont(holdFont);

  }


  public void drawAggregate(Graphics graphics) {

    double aggregateX = 0;
    double aggregateY = 0;

    if (graphics == null) {
      return;
    }

    if (aggregateAttribute.length() == 0) {
      return;
    }

    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;

    // Draw the condition name and connection line
    aggregateX = x3 + getWidth()/2;
    aggregateY = getY() - getHeight();


    // draw the node
    Shape aggregateNode = new Ellipse2D.Double(aggregateX,
                                               aggregateY,
Operator.NODE_RADIUS,NODE_RADIUS);

    g.draw(aggregateNode);

    // Draw the conection between two nodes
    g.drawLine((int) (x3 + Operator.NODE_RADIUS/2),
               (int) y3,
               (int) (aggregateX + Operator.NODE_RADIUS/2),
               (int) (aggregateY + Operator.NODE_RADIUS));

    // Calculate the length of the name
```

```
  Font holdFont = graphics.getFont();
  graphics.setFont(new Font(holdFont.getName(),
                            holdFont.getStyle(),
                            holdFont.getSize() - 2));
  FontMetrics fm = graphics.getFontMetrics();

  // System.out.println("Font size = " + holdFont.getSize());

  double nameWidth = fm.stringWidth(aggregateAttribute) + 2;
  double nameHeight = fm.getHeight() + 6;

  // draw a straight line
  g.drawLine((int) (aggregateX - (nameWidth/2)),(int) aggregateY,
             (int) (aggregateX + (nameWidth/2)),(int) aggregateY);

  // draw the aggregate name
  g.drawString(aggregateAttribute,
               (int) (aggregateX - (nameWidth/2) + 1),
               (int) (aggregateY - 3));


 // after all, change the font back to original
 graphics.setFont(holdFont);

}

// Action area
public String buildQuery() {
  StringBuffer qry = new StringBuffer("select distinct ");
  if (groupAttributes.length() != 0) {
    qry.append(groupAttributes);
  }
  if (aggregateAttribute.length() != 0) {
    if (groupAttributes.length() != 0) {
      qry.append(", ");
    }
    // MAX keywork of the aggregate attributes
    qry.append("max(").append(aggregateAttribute).append(")");
  } else {
    return (new String(""));
  }

  qry.append(" from ");
  // Check if the relation is operator object in the collection
  if (isInCollection(relation)) {
    qry.append(" (" );
    Operator op = (Operator) getObject(relation);
    qry.append(op.buildQuery());
    qry.append(")");
  } else {
    qry.append(relation);
  }
  // group by
  if (groupAttributes.length() > 0) {
    qry.append(" group by ").append(groupAttributes);
  }
```

```
    return qry.toString();
  }


  // Implement Externalizable interface - write
  public void writeExternal(ObjectOutput out) throws IOException {
    // Call the super class to save the common data
    super.writeExternal(out);

    // Write the data belong to this operator
    out.writeObject(relation);
    out.writeObject(groupAttributes);
    out.writeObject(aggregateAttribute);

  }


  // Implement Externalizable interface - read
  public void readExternal(ObjectInput in) {
    // call the super class to get the common data
    super.readExternal(in);

    // read the data belongs to this operator
    try {
      relation = (String) in.readObject();
      groupAttributes = (String) in.readObject();
      aggregateAttribute = (String) in.readObject();
   } catch (Exception e) {
      System.err.println(e);
   }
  }
}
```

17. OperatorGroupmin.java

```
/*
 * Author: Ron Chen
 * File: OperatorGroupmin.java
 *
 * A Class for groupmin operator for DFQL
 *
*/

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import java.io.*;
import java.util.*;


public class OperatorGroupmin extends Operator
{

  final static String OPERATOR_TYPE = "groupmin";
```

```
  String[] labels = {"Name", "Position X", "Position Y",
                     "Relation", "Grouping Attributes", "Aggregate
Attribute"};
  JLabel[] propertyLabels = new JLabel[labels.length];
  JTextField[] propertyTextFields = new JTextField[labels.length];

  // ------ Property ----------------
  private String relation="";
  private String groupAttributes="";
  private String aggregateAttribute="";

  // ------ end of property ----------
  public OperatorGroupmin() {
    super(OPERATOR_TYPE);
  }

  public OperatorGroupmin(Vector vRef) {
    super(OPERATOR_TYPE, vRef);
  }

  // -------- Property get/set ------------
  public String getRelation() {
    return relation;
  }

  public void setRelation(String sRelation) {
    relation = new String(sRelation);
  }

  public String getGroupAttributes() {
    return groupAttributes;
  }

  public void setGroupAttributes(String sGroupAttributes) {
    groupAttributes = new String(sGroupAttributes);
  }

  public String getAggregateAttribute() {
    return aggregateAttribute;
  }

  public void setAggregateAttribute(String sAggregateAttribute) {
    if (!(sAggregateAttribute == null)) {
      aggregateAttribute = new String(sAggregateAttribute);
    }
  }
  // --------- end of property get/set -------

  // This method is specific used by OperatorUsre.java class
  // when the input node node link to the InputBar Node
  // when the buildQuery() method, the value of each link node
  // will pass to the current operator.  Once the query is built,
  // the input node value will be reset to the original value
  public void setInputNodeValue(int nNode, String value) {
    // empty body - should extended by each child class
    switch (nNode) {
      case 1:          // relation
        setRelation(value);
```

```
      break;
    case 2:          // group attributes
      setGroupAttributes(value);
      break;
    case 3:          // aggregate attribute
      setAggregateAttribute(value);
      break;
    default:
  }
}

// mouse events on this class
public void mouseClicked(MouseEvent e) {
  // call the base class event
  super.mouseClicked(e);

  // if this is right mouse clicked, then popup
  // property window
  if (!isInBound(e.getX(), e.getY())) {
    return ;
  }

  // Check if this operator should be redrawn
  // this condition also prevents the property window
  // pop up twice.
  if (isDirty()) {
    return;
  }

  if ( e.getModifiers() == e.BUTTON3_MASK) {

    for (int i=0; i<labels.length; ++i) {
      String fieldText="";

      // Assigned the filed name
      propertyLabels[i] = new JLabel(labels[i]);
      // Get the filed information
      switch (i) {
        case 0:            // name
          fieldText = new String(getName());
          break;
        case 1:            // position X
          fieldText = String.valueOf(getX());
          break;
        case 2:            // position Y
          fieldText = String.valueOf(getY());
          break;
        case 3:            // relation
          fieldText = relation;
          break;
        case 4:            // group attributes
          fieldText = groupAttributes;
          break;
        case 5:            // aggregate attribute
          fieldText = aggregateAttribute;
          break;
        default:
      }
```

```
        propertyTextFields[i] = new JTextField(new String(fieldText));
      }

      // System.out.println("popup the property window");

      PropertyWindow propertyWindow = new
PropertyWindow(FrameMain.parent,
                                                    propertyLabels,

propertyTextFields);

      if (propertyWindow.propertyOption == propertyWindow.OK) {
        // Save the changes
        for (int i=0; i<labels.length; ++i) {

          String fieldText = propertyTextFields[i].getText();

          // Get the field inforamtion from the text field
          switch (i) {
            case 0:           // name
              setName(fieldText);
              break;
            case 1:           // position X
              setX((new Double(fieldText)).doubleValue());
              break;
            case 2:           // position Y
              setY((new Double(fieldText)).doubleValue());
              break;
            case 3:           // relation
              setRelation(fieldText);
              break;
            case 4:           // group attributes
              setGroupAttributes(fieldText);
              break;
            case 5:           // aggregate attributes
              setAggregateAttribute(fieldText);
              break;
            default:
          }

        }

      }
      propertyWindow.dispose();

      // this operator should be redrawn
      setDirty(true);
    }

  }
  public void draw(Graphics g) {
    if (g == null) {
      return;
    }

    // Draw the main body
    super.draw(g);
    // Draw the nodes
```

```
    super.drawThreeInputNodes(g);
    // Draw the relation name
    drawRelation(g);
    // Draw the group attributes name
    drawGroupAttributes(g);

    // Draw the Aggregate Attributes string
    drawAggregate(g);

    // now, the operator should not be redrawn again
    super.setDirty(false);

  }

  public void drawRelation(Graphics graphics) {

    double relationX = 0;
    double relationY = 0;

    if (graphics == null) {
      return;
    }

    if (relation.length() == 0) {
      return;
    }


    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;

    // Make sure this oper
    if (isInCollection(relation)) {
      // Draw the connection line to output node (ox, oy) on relation
      // System.out.println(relation + " is in the collection");
      Operator op = (Operator) getObject(relation);

      g.drawLine((int) (op.ox + Operator.NODE_RADIUS/2),
                 (int) (op.oy + Operator.NODE_RADIUS),
                 (int) (this.x1 + Operator.NODE_RADIUS/2),
                 (int) (this.y1));

    } else {
      // Draw the operator name and connection line
      //System.out.println(relation + " is not in the collection");

      relationX = x1 - getWidth()/2;
      relationY = getY() - getHeight();


      // draw the node
      Shape relationNode = new Ellipse2D.Double(relationX,
                                                relationY,
Operator.NODE_RADIUS,NODE_RADIUS);

      g.draw(relationNode);
```

```
    // Draw the conection between two nodes
    g.drawLine((int) (x1 + Operator.NODE_RADIUS/2),
               (int) y1,
               (int) (relationX + Operator.NODE_RADIUS/2),
               (int) (relationY + Operator.NODE_RADIUS));

    // Calculate the length of the name

    Font holdFont = graphics.getFont();
    graphics.setFont(new Font(holdFont.getName(),
                              holdFont.getStyle(),
                              holdFont.getSize() - 2));
    FontMetrics fm = graphics.getFontMetrics();

    // System.out.println("Font size = " + holdFont.getSize());

    double nameWidth = fm.stringWidth(relation) + 2;
    double nameHeight = fm.getHeight() + 6;

    // draw a straight line
    g.drawLine((int) (relationX - (nameWidth/2)),(int) relationY,
               (int) (relationX + (nameWidth/2)),(int) relationY);

    // draw the relation name
    g.drawString(relation,
                 (int) (relationX - (nameWidth/2) + 1),
                 (int) (relationY - 3));


   // after all, change the font back to original
   graphics.setFont(holdFont);

  }


}

public void drawGroupAttributes(Graphics graphics) {

  double groupX = 0;
  double groupY = 0;


  if (graphics == null) {
    return;
  }

  if (groupAttributes.length() == 0) {
    return;
  }


  // Use the Graphics2D object
  Graphics2D g = (Graphics2D) graphics;


  groupX = x2;
  groupY = y2 - getHeight() * 2;   // twice's high
```

```
    // draw the node
    Shape groupNode = new Ellipse2D.Double(groupX,
                                           groupY,
Operator.NODE_RADIUS,NODE_RADIUS);

    g.draw(groupNode);

    // Draw the conection between two nodes
    g.drawLine((int) (x2 + Operator.NODE_RADIUS/2),
               (int) y2,
               (int) (groupX + Operator.NODE_RADIUS/2),
               (int) (groupY + Operator.NODE_RADIUS));

    // Calculate the length of the name

    Font holdFont = graphics.getFont();
    graphics.setFont(new Font(holdFont.getName(),
                              holdFont.getStyle(),
                              holdFont.getSize() - 2));
    FontMetrics fm = graphics.getFontMetrics();

    // System.out.println("Font size = " + holdFont.getSize());

    double nameWidth = fm.stringWidth(groupAttributes) + 2;
    double nameHeight = fm.getHeight() + 6;

    // draw a straight line
    g.drawLine((int) (groupX - (nameWidth/2)),(int) groupY,
               (int) (groupX + (nameWidth/2)),(int) groupY);

    // draw the relation name
    g.drawString(groupAttributes,
                 (int) (groupX - (nameWidth/2) + 1),
                 (int) (groupY - 3));


   // after all, change the font back to original
   graphics.setFont(holdFont);

  }


  public void drawAggregate(Graphics graphics) {

    double aggregateX = 0;
    double aggregateY = 0;

    if (graphics == null) {
      return;
    }

    if (aggregateAttribute.length() == 0) {
      return;
    }
```

```
    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;

    // Draw the condition name and connection line
    aggregateX = x3 + getWidth()/2;
    aggregateY = getY() - getHeight();


    // draw the node
    Shape aggregateNode = new Ellipse2D.Double(aggregateX,
                                               aggregateY,

Operator.NODE_RADIUS,NODE_RADIUS);

    g.draw(aggregateNode);

    // Draw the conection between two nodes
    g.drawLine((int) (x3 + Operator.NODE_RADIUS/2),
               (int) y3,
               (int) (aggregateX + Operator.NODE_RADIUS/2),
               (int) (aggregateY + Operator.NODE_RADIUS));

    // Calculate the length of the name

    Font holdFont = graphics.getFont();
    graphics.setFont(new Font(holdFont.getName(),
                              holdFont.getStyle(),
                              holdFont.getSize() - 2));
    FontMetrics fm = graphics.getFontMetrics();

    // System.out.println("Font size = " + holdFont.getSize());

    double nameWidth = fm.stringWidth(aggregateAttribute) + 2;
    double nameHeight = fm.getHeight() + 6;

    // draw a straight line
    g.drawLine((int) (aggregateX - (nameWidth/2)),(int) aggregateY,
               (int) (aggregateX + (nameWidth/2)),(int) aggregateY);

    // draw the aggregate name
    g.drawString(aggregateAttribute,
                 (int) (aggregateX - (nameWidth/2) + 1),
                 (int) (aggregateY - 3));


   // after all, change the font back to original
   graphics.setFont(holdFont);

  }

  // Action area
  public String buildQuery() {
    StringBuffer qry = new StringBuffer("select distinct ");
    if (groupAttributes.length() != 0) {
      qry.append(groupAttributes);
    }
    if (aggregateAttribute.length() != 0) {
      if (groupAttributes.length() != 0) {
```

```
        qry.append(", ");
      }
      // MAX keywork of the aggregate attributes
      qry.append("min(").append(aggregateAttribute).append(")");
    } else {
      return (new String(""));
    }

    qry.append(" from ");
    // Check if the relation is operator object in the collection
    if (isInCollection(relation)) {
      qry.append(" (" );
      Operator op = (Operator) getObject(relation);
      qry.append(op.buildQuery());
      qry.append(")");
    } else {
      qry.append(relation);
    }
    // group by
    if (groupAttributes.length() > 0) {
      qry.append(" group by ").append(groupAttributes);
    }

    return qry.toString();
  }


  // Implement Externalizable interface - write
  public void writeExternal(ObjectOutput out) throws IOException {
    // Call the super class to save the common data
    super.writeExternal(out);

    // Write the data belong to this operator
    out.writeObject(relation);
    out.writeObject(groupAttributes);
    out.writeObject(aggregateAttribute);

  }


  // Implement Externalizable interface - read
  public void readExternal(ObjectInput in) {
    // call the super class to get the common data
    super.readExternal(in);

    // read the data belongs to this operator
    try {
      relation = (String) in.readObject();
      groupAttributes = (String) in.readObject();
      aggregateAttribute = (String) in.readObject();
   } catch (Exception e) {
      System.err.println(e);
   }
  }
}
```

18. OperatorGroupNsatisfy.java

```
/*
 * Author: Ron Chen
 * File: OperatorGroupNsatisfy.java
 *
 * A Class for groupNsatisfy operator for DFQL
 * This is very similar operator as OperatorGroupALLsatisfy.java class
 * except return specify N records
 *
*/

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import java.io.*;
import java.util.*;


public class OperatorGroupNsatisfy extends Operator
{

  final static String OPERATOR_TYPE = "groupNsatisfy";
  String[] labels = {"Name", "Position X", "Position Y",
                     "Relation", "Grouping Attributes", "Condition",
                     "Row Number"};
  JLabel[] propertyLabels = new JLabel[labels.length];
  JTextField[] propertyTextFields = new JTextField[labels.length];

  // ------ Property -----------------
  private String relation="";
  private String groupAttributes="";
  private String condition="";
  private String rowNum = "";
  // ------ end of property ----------

  public OperatorGroupNsatisfy() {
    super(OPERATOR_TYPE);
  }


  public OperatorGroupNsatisfy(Vector vRef) {
    super(OPERATOR_TYPE, vRef);
  }

  // -------- Property get/set ------------
  public String getRelation() {
    return relation;
  }

  public void setRelation(String sRelation) {
    relation = new String(sRelation);
  }

  public String getGroupAttributes() {
    return groupAttributes;
  }
```

```
public void setGroupAttributes(String sGroupAttributes) {
  groupAttributes = new String(sGroupAttributes);
}

public String getCondition() {
  return condition;
}

public void setCondition(String sCondition) {
  if (!(sCondition == null)) {
    condition = new String(sCondition);
  }
}

public String getNumber() {
  return rowNum;
}

public void setNumber(String sNum) {
  if (!(sNum == null)) {
    rowNum = new String(sNum);
  }
}
// --------- end of property get/set -------

// This method is specific used by OperatorUsre.java class
// when the input node node link to the InputBar Node
// when the buildQuery() method, the value of each link node
// will pass to the current operator.  Once the query is built,
// the input node value will be reset to the original value
public void setInputNodeValue(int nNode, String value) {
  // empty body - should extended by each child class
  switch (nNode) {
    case 1:          // relation
      setRelation(value);
      break;
    case 2:          // group attributes
      setGroupAttributes(value);
      break;
    case 3:          // condition
      setCondition(value);
      break;
    case 4:          // number
      setNumber(value);
      break;
    default:
  }
}
// mouse events on this class
public void mouseClicked(MouseEvent e) {
  // call the base class event
  super.mouseClicked(e);

  // if this is right mouse clicked, then popup
  // property window
  if (!isInBound(e.getX(), e.getY())) {
    return ;
```

```
    }

    // Check if this operator should be redrawn
    // this condition also prevents the property window
    // pop up twice.
    if (isDirty()) {
      return;
    }

    if ( e.getModifiers() == e.BUTTON3_MASK) {

      for (int i=0; i<labels.length; ++i) {
        String fieldText="";

        // Assigned the filed name
        propertyLabels[i] = new JLabel(labels[i]);
        // Get the filed information
        switch (i) {
          case 0:           // name
            fieldText = new String(getName());
            break;
          case 1:           // position X
            fieldText = String.valueOf(getX());
            break;
          case 2:           // position Y
            fieldText = String.valueOf(getY());
            break;
          case 3:           // relation
            fieldText = relation;
            break;
          case 4:           // group attributes
            fieldText = groupAttributes;
            break;
          case 5:           // condition
            fieldText = condition;
            break;
          case 6:           // Row Number
            fieldText = rowNum;
            break;
          default:
        }
        propertyTextFields[i] = new JTextField(new String(fieldText));
      }

      // System.out.println("popup the property window");

      PropertyWindow propertyWindow = new
PropertyWindow(FrameMain.parent,
                                                    propertyLabels,
propertyTextFields);

      if (propertyWindow.propertyOption == propertyWindow.OK) {
        // Save the changes
        for (int i=0; i<labels.length; ++i) {

          String fieldText = propertyTextFields[i].getText();
```

```
            // Get the field inforamtion from the text field
            switch (i) {
              case 0:           // name
                setName(fieldText);
                break;
              case 1:           // position X
                setX((new Double(fieldText)).doubleValue());
                break;
              case 2:           // position Y
                setY((new Double(fieldText)).doubleValue());
                break;
              case 3:           // relation
                setRelation(fieldText);
                break;
              case 4:           // group attributes
                setGroupAttributes(fieldText);
                break;
              case 5:           // condition
                setCondition(fieldText);
                break;
              case 6:           // Row Number
                setNumber(fieldText);
                break;
              default:
            }

          }

        }
        propertyWindow.dispose();

        // this operator should be redrawn
        setDirty(true);
      }

    }

    public void draw(Graphics g) {
      if (g == null) {
        return;
      }

      // Draw the main body
      super.draw(g);
      // Draw the nodes
      super.drawFourInputNodes(g);
      // Draw the relation name
      drawRelation(g);
      // Draw the group attributes name
      drawGroupAttributes(g);
      // Draw the Count Attributes string
      drawCondition(g);
      // Draw the number
      drawNumber(g);

      // now, the operator should not be redrawn again
      super.setDirty(false);
```

```
  }

  public void drawRelation(Graphics graphics) {

    double relationX = 0;
    double relationY = 0;

    if (graphics == null) {
      return;
    }

    if (relation.length() == 0) {
      return;
    }


    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;

    // Make sure this oper
    if (isInCollection(relation)) {
      // Draw the connection line to output node (ox, oy) on relation
      // System.out.println(relation + " is in the collection");
      Operator op = (Operator) getObject(relation);

      g.drawLine((int) (op.ox + Operator.NODE_RADIUS/2),
                 (int) (op.oy + Operator.NODE_RADIUS),
                 (int) (this.x1 + Operator.NODE_RADIUS/2),
                 (int) (this.y1));

    } else {
      // Draw the operator name and connection line
      //System.out.println(relation + " is not in the collection");

      relationX = x1 - getWidth()/2;
      relationY = getY() - getHeight();


      // draw the node
      Shape relationNode = new Ellipse2D.Double(relationX,
                                                relationY,

Operator.NODE_RADIUS,NODE_RADIUS);

      g.draw(relationNode);

      // Draw the conection between two nodes
      g.drawLine((int) (x1 + Operator.NODE_RADIUS/2),
                 (int) y1,
                 (int) (relationX + Operator.NODE_RADIUS/2),
                 (int) (relationY + Operator.NODE_RADIUS));

      // Calculate the length of the name

      Font holdFont = graphics.getFont();
      graphics.setFont(new Font(holdFont.getName(),
                                holdFont.getStyle(),
                                holdFont.getSize() - 2));
```

```
      FontMetrics fm = graphics.getFontMetrics();

      // System.out.println("Font size = " + holdFont.getSize());

      double nameWidth = fm.stringWidth(relation) + 2;
      double nameHeight = fm.getHeight() + 6;

      // draw a straight line
      g.drawLine((int) (relationX - (nameWidth/2)),(int) relationY,
                 (int) (relationX + (nameWidth/2)),(int) relationY);

      // draw the relation name
      g.drawString(relation,
                   (int) (relationX - (nameWidth/2) + 1),
                   (int) (relationY - 3));


     // after all, change the font back to original
     graphics.setFont(holdFont);

    }


  }

  public void drawGroupAttributes(Graphics graphics) {

    double groupX = 0;
    double groupY = 0;


    if (graphics == null) {
      return;
    }

    if (groupAttributes.length() == 0) {
      return;
    }


    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;


    groupX = x2;
    groupY = y2 - getHeight() * 2;    // 2 times high


    // draw the node
    Shape groupNode = new Ellipse2D.Double(groupX,
                                           groupY,
Operator.NODE_RADIUS,NODE_RADIUS);

    g.draw(groupNode);

    // Draw the conection between two nodes
    g.drawLine((int) (x2 + Operator.NODE_RADIUS/2),
```

```
            (int) y2,
            (int) (groupX + Operator.NODE_RADIUS/2),
            (int) (groupY + Operator.NODE_RADIUS));

  // Calculate the length of the name

  Font holdFont = graphics.getFont();
  graphics.setFont(new Font(holdFont.getName(),
                            holdFont.getStyle(),
                            holdFont.getSize() - 2));
  FontMetrics fm = graphics.getFontMetrics();

  // System.out.println("Font size = " + holdFont.getSize());

  double nameWidth = fm.stringWidth(groupAttributes) + 2;
  double nameHeight = fm.getHeight() + 6;

  // draw a straight line
  g.drawLine((int) (groupX - (nameWidth/2)),(int) groupY,
             (int) (groupX + (nameWidth/2)),(int) groupY);

  // draw the relation name
  g.drawString(groupAttributes,
               (int) (groupX - (nameWidth/2) + 1),
               (int) (groupY - 3));


 // after all, change the font back to original
 graphics.setFont(holdFont);

}


public void drawCondition(Graphics graphics) {

  double conditionX = 0;
  double conditionY = 0;

  if (graphics == null) {
    return;
  }

  if (condition.length() == 0) {
    return;
  }

  // Use the Graphics2D object
  Graphics2D g = (Graphics2D) graphics;

  // Draw the condition name and connection line
  conditionX = x3;
  conditionY = getY() - getHeight()*1.5;      // 1.5 times high


  // draw the node
  Shape conditionNode = new Ellipse2D.Double(conditionX,
                                             conditionY,
```

```
Operator.NODE_RADIUS,NODE_RADIUS);

    g.draw(conditionNode);

    // Draw the conection between two nodes
    g.drawLine((int) (x3 + Operator.NODE_RADIUS/2),
               (int) y3,
               (int) (conditionX + Operator.NODE_RADIUS/2),
               (int) (conditionY + Operator.NODE_RADIUS));

    // Calculate the length of the name

    Font holdFont = graphics.getFont();
    graphics.setFont(new Font(holdFont.getName(),
                              holdFont.getStyle(),
                              holdFont.getSize() - 2));
    FontMetrics fm = graphics.getFontMetrics();

    // System.out.println("Font size = " + holdFont.getSize());

    double nameWidth = fm.stringWidth(condition) + 2;
    double nameHeight = fm.getHeight() + 6;

    // draw a straight line
    g.drawLine((int) (conditionX - (nameWidth/2)),(int) conditionY,
               (int) (conditionX + (nameWidth/2)),(int) conditionY);

    // draw the relation name
    g.drawString(condition,
                 (int) (conditionX - (nameWidth/2) + 1),
                 (int) (conditionY - 3));


   // after all, change the font back to original
   graphics.setFont(holdFont);

  }


  public void drawNumber(Graphics graphics) {

    double numberX = 0;
    double numberY = 0;

    if (graphics == null) {
      return;
    }

    if (rowNum.length() == 0) {
      return;
    }

    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;

    // Draw the condition name and connection line
    numberX = x4 + getWidth()/2;
```

```
    numberY = getY() - getHeight();


    // draw the node
    Shape numberNode = new Ellipse2D.Double(numberX,
                                            numberY,

Operator.NODE_RADIUS,NODE_RADIUS);

    g.draw(numberNode);

    // Draw the conection between two nodes
    g.drawLine((int) (x4 + Operator.NODE_RADIUS/2),
               (int) y4,
               (int) (numberX + Operator.NODE_RADIUS/2),
               (int) (numberY + Operator.NODE_RADIUS));

    // Calculate the length of the name

    Font holdFont = graphics.getFont();
    graphics.setFont(new Font(holdFont.getName(),
                              holdFont.getStyle(),
                              holdFont.getSize() - 2));
    FontMetrics fm = graphics.getFontMetrics();

    // System.out.println("Font size = " + holdFont.getSize());

    double nameWidth = fm.stringWidth(rowNum) + 2;
    double nameHeight = fm.getHeight() + 6;

    // draw a straight line
    g.drawLine((int) (numberX - (nameWidth/2)),(int) numberY,
               (int) (numberX + (nameWidth/2)),(int) numberY);

    // draw the relation name
    g.drawString(rowNum,
                 (int) (numberX - (nameWidth/2) + 1),
                 (int) (numberY - 3));


   // after all, change the font back to original
   graphics.setFont(holdFont);

  }

  // Action area
  public String buildQuery() {
    StringBuffer qry = new StringBuffer("select distinct ");
    if (groupAttributes.length() != 0) {
      qry.append(groupAttributes);
    }
    else {
      return (new String(""));
    }

    qry.append(" from ");
    // Check if the relation is operator object in the collection
    if (isInCollection(relation)) {
```

```
        qry.append(" (" );
        Operator op = (Operator) getObject(relation);
        qry.append(op.buildQuery());
        qry.append(")");
      } else {
        qry.append(relation);
      }
      // check condition
      if (condition.length() > 0) {
        qry.append(" where ").append(condition);

      }
      // check the number
      if (rowNum.length() > 0) {
        qry.append(" and rownum ").append(rowNum);
      }
      // group by
      if (groupAttributes.length() > 0) {
        qry.append(" group by ").append(groupAttributes);
      }

      return qry.toString();
    }


    // Implement Externalizable interface - write
    public void writeExternal(ObjectOutput out) throws IOException {
      // Call the super class to save the common data
      super.writeExternal(out);

      // Write the data belong to this operator
      out.writeObject(relation);
      out.writeObject(groupAttributes);
      out.writeObject(condition);
      out.writeObject(rowNum);
    }


    // Implement Externalizable interface - read
    public void readExternal(ObjectInput in) {
      // call the super class to get the common data
      super.readExternal(in);

      // read the data belongs to this operator
      try {
        relation = (String) in.readObject();
        groupAttributes = (String) in.readObject();
        condition = (String) in.readObject();
        rowNum = (String) in.readObject();
     } catch (Exception e) {
        System.err.println(e);
     }
    }
}
```

19. OperatorIntersect.java

```
/*
 * Author: Ron Chen
 * File: OperatorIntersect.java
 *
 * A Class for intersect operator for DFQL
 *
*/

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import java.io.*;
import java.util.*;


public class OperatorIntersect extends Operator
{

  final static String OPERATOR_TYPE = "intersect";

  String[] labels = {"Name", "Position X", "Position Y",
                     "Relation 1", "Relation 2"};
  JLabel[] propertyLabels = new JLabel[labels.length];
  JTextField[] propertyTextFields = new JTextField[labels.length];

  // ------ Property -----------------
  private String relation1="";
  private String relation2="";

  // ------ end of property ----------

  public OperatorIntersect() {
    super(OPERATOR_TYPE);
  }

  public OperatorIntersect(Vector vRef) {
    super(OPERATOR_TYPE, vRef);
  }

  // -------- Property get/set ------------
  public String getRelation1() {
    return relation1;
  }

  public void setRelation1(String sRelation) {
    relation1 = new String(sRelation);
  }

  public String getRelation2() {
    return relation2;
  }

  public void setRelation2(String sRelation) {
    relation2 = new String(sRelation);
  }
```

```
// --------- end of property get/set -------

// This method is specific used by OperatorUsre.java class
// when the input node node link to the InputBar Node
// when the buildQuery() method, the value of each link node
// will pass to the current operator.  Once the query is built,
// the input node value will be reset to the original value
public void setInputNodeValue(int nNode, String value) {
  // empty body - should extended by each child class
  switch (nNode) {
    case 1:          // relation 1
      setRelation1(value);
      break;
    case 2:          // relation 2
      setRelation2(value);
      break;
    default:
  }
}

// mouse events on this class
public void mouseClicked(MouseEvent e) {
  // call the base class event
  super.mouseClicked(e);

  // if this is right mouse clicked, then popup
  // property window
  if (!isInBound(e.getX(), e.getY())) {
    return ;
  }

  // Check if this operator should be redrawn
  // this condition also prevents the property window
  // pop up twice.
  if (isDirty()) {
    return;
  }

  if ( e.getModifiers() == e.BUTTON3_MASK) {
    for (int i=0; i<labels.length; ++i) {
      String fieldText="";

      // Assigned the filed name
      propertyLabels[i] = new JLabel(labels[i]);
      // Get the filed information
      switch (i) {
        case 0:            // name
          fieldText = new String(getName());
          break;
        case 1:            // position X
          fieldText = String.valueOf(getX());
          break;
        case 2:            // position Y
          fieldText = String.valueOf(getY());
          break;
        case 3:            // relation 1
          fieldText = relation1;
```

```
            break;
          case 4:           // relation 2
            fieldText = relation2;
            break;
          default:
        }
        propertyTextFields[i] = new JTextField(new String(fieldText));
      }

      // System.out.println("popup the property window");

      PropertyWindow propertyWindow = new
PropertyWindow(FrameMain.parent,
                                                    propertyLabels,

propertyTextFields);

      if (propertyWindow.propertyOption == propertyWindow.OK) {
        // Save the changes
        for (int i=0; i<labels.length; ++i) {

          String fieldText = propertyTextFields[i].getText();

          // Get the field inforamtion from the text field
          switch (i) {
            case 0:           // name
              setName(fieldText);
              break;
            case 1:           // position X
              setX((new Double(fieldText)).doubleValue());
              break;
            case 2:           // position Y
              setY((new Double(fieldText)).doubleValue());
              break;
            case 3:           // relation 1
              setRelation1(fieldText);
              break;
            case 4:           // relation 2
              setRelation2(fieldText);
              break;
            default:
          }

        }

      }
      propertyWindow.dispose();

      // this operator should be redrawn
      setDirty(true);
    }

  }

  public void draw(Graphics g) {
    if (g == null) {
      System.out.println("graphics is null on OperatorIntersect-
>draw()");
```

```
      return;
    }

    // Draw the main body
    super.draw(g);
    // Draw the nodes
    super.drawTwoInputNodes(g);

    // Draw the relation 1 name
    drawRelation1(g);
    // Draw the relation 2 name
    drawRelation2(g);

    // now, the operator should not be redrawn again
    super.setDirty(false);

  }

  public void drawRelation1(Graphics graphics) {

    double relationX = 0;
    double relationY = 0;
    String relation = "";

    relation = relation1;

    if (graphics == null) {
      return;
    }

    if (relation.length() == 0) {
      return;
    }

    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;

    // Make sure this oper
    if (isInCollection(relation)) {
      // Draw the connection line to output node (ox, oy) on relation
      // System.out.println(relation + " is in the collection");
      Operator op = (Operator) getObject(relation);

      g.drawLine((int) (op.ox + Operator.NODE_RADIUS/2),
                 (int) (op.oy + Operator.NODE_RADIUS),
                 (int) (this.x1 + Operator.NODE_RADIUS/2),
                 (int) (this.y1));

    } else {
      // Draw the operator name and connection line
      //System.out.println(relation + " is not in the collection");

      relationX = getX();
      relationY = getY() - getHeight();


      // draw the node
      Shape relationNode = new Ellipse2D.Double(relationX,
```

```
                                                    relationY,
Operator.NODE_RADIUS,NODE_RADIUS);

      g.draw(relationNode);

      // Draw the conection between two nodes
      g.drawLine((int) (x1 + Operator.NODE_RADIUS/2),
                 (int) y1,
                 (int) (relationX + Operator.NODE_RADIUS/2),
                 (int) (relationY + Operator.NODE_RADIUS));

      // Calculate the length of the name

      Font holdFont = graphics.getFont();
      graphics.setFont(new Font(holdFont.getName(),
                                holdFont.getStyle(),
                                holdFont.getSize() - 2));
      FontMetrics fm = graphics.getFontMetrics();

      // System.out.println("Font size = " + holdFont.getSize());

      double nameWidth = fm.stringWidth(relation) + 2;
      double nameHeight = fm.getHeight() + 6;

      // draw a straight line
      g.drawLine((int) (relationX - (nameWidth/2)),(int) relationY,
                 (int) (relationX + (nameWidth/2)),(int) relationY);

      // draw the relation name
      g.drawString(relation,
                   (int) (relationX - (nameWidth/2) + 1),
                   (int) (relationY - 3));


     // after all, change the font back to original
     graphics.setFont(holdFont);

    }


  }

  public void drawRelation2(Graphics graphics) {

    double relationX = 0;
    double relationY = 0;

    String relation = "";

    relation = relation2;

    if (graphics == null) {
      return;
    }

    if (relation.length() == 0) {
      return;
```

```
    }

    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;

    // Make sure this oper
    if (isInCollection(relation)) {
      // Draw the connection line to output node (ox, oy) on relation
      // System.out.println(relation + " is in the collection");
      Operator op = (Operator) getObject(relation);

      g.drawLine((int) (op.ox + Operator.NODE_RADIUS/2),
                 (int) (op.oy + Operator.NODE_RADIUS),
                 (int) (this.x2 + Operator.NODE_RADIUS/2),
                 (int) (this.y2));

    } else {
      // Draw the operator name and connection line
      //System.out.println(relation + " is not in the collection");

      relationX = getX() + getWidth();
      relationY = getY() - getHeight();


      // draw the node
      Shape relationNode = new Ellipse2D.Double(relationX,
                                                relationY,

Operator.NODE_RADIUS,NODE_RADIUS);

      g.draw(relationNode);

      // Draw the conection between two nodes
      g.drawLine((int) (x2 + Operator.NODE_RADIUS/2),
                 (int) y2,
                 (int) (relationX + Operator.NODE_RADIUS/2),
                 (int) (relationY + Operator.NODE_RADIUS));

      // Calculate the length of the name

      Font holdFont = graphics.getFont();
      graphics.setFont(new Font(holdFont.getName(),
                                holdFont.getStyle(),
                                holdFont.getSize() - 2));
      FontMetrics fm = graphics.getFontMetrics();

      // System.out.println("Font size = " + holdFont.getSize());

      double nameWidth = fm.stringWidth(relation) + 2;
      double nameHeight = fm.getHeight() + 6;

      // draw a straight line
      g.drawLine((int) (relationX - (nameWidth/2)),(int) relationY,
                 (int) (relationX + (nameWidth/2)),(int) relationY);

      // draw the relation name
      g.drawString(relation,
                   (int) (relationX - (nameWidth/2) + 1),
```

```
                (int) (relationY - 3));


    // after all, change the font back to original
    graphics.setFont(holdFont);

  }
}

// Action area
public String buildQuery() {
  StringBuffer qry = new StringBuffer("select distinct * from ");
  // Check if the relation is operator object in the collection
  if (isInCollection(relation1)) {
    qry.append(" (" );
    Operator op = (Operator) getObject(relation1);
    qry.append(op.buildQuery());
    qry.append(")");
  } else {
    qry.append(relation1);
  }
  qry.append(" intersect ");
  qry.append("select distinct * from ");
  if (isInCollection(relation2)) {
    qry.append(" (" );
    Operator op = (Operator) getObject(relation2);
    qry.append(op.buildQuery());
    qry.append(")");
  } else {
    qry.append(relation2);
  }

  return qry.toString();
}


// Implement Externalizable interface - write
public void writeExternal(ObjectOutput out) throws IOException {
  // Call the super class to save the common data
  super.writeExternal(out);

  // Write the data belong to this operator
  out.writeObject(relation1);
  out.writeObject(relation2);

}


// Implement Externalizable interface - read
public void readExternal(ObjectInput in) {
  // call the super class to get the common data
  super.readExternal(in);

  // read the data belongs to this operator
  try {
    relation1 = (String) in.readObject();
    relation2 = (String) in.readObject();
 } catch (Exception e) {
```

```
        System.err.println(e);
      }
    }
}
```

20. OperatorJoin.java

```
/*
 * Author: Ron Chen
 * File: OperatorJoin.java
 *
 * A Class for join operator for DFQL
 *
*/

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import java.io.*;
import java.util.*;


public class OperatorJoin extends Operator
{

  final static String OPERATOR_TYPE = "join";

  String[] labels = {"Name", "Position X", "Position Y",
                     "Relation 1", "Relation 2", "Condition"};
  JLabel[] propertyLabels = new JLabel[labels.length];
  JTextField[] propertyTextFields = new JTextField[labels.length];

  // ------ Property -----------------
  private String relation1="";
  private String relation2="";
  private String condition="";

  // ------ end of property ----------

  public OperatorJoin() {
    super(OPERATOR_TYPE);
  }

  public OperatorJoin(Vector vRef) {
    super(OPERATOR_TYPE, vRef);
  }

  // -------- Property get/set ------------
  public String getRelation1() {
    return relation1;
  }

  public void setRelation1(String sRelation1) {
    relation1 = new String(sRelation1);
```

```
}

public String getRelation2() {
  return relation2;
}

public void setRelation2(String sRelation2) {
  relation2 = new String(sRelation2);
}

public String getCondition() {
  return condition;
}

public void setCondition(String sCondition) {
  if (!(sCondition == null)) {
    condition = new String(sCondition);
  }
}
// --------- end of property get/set -------

// This method is specific used by OperatorUsre.java class
// when the input node node link to the InputBar Node
// when the buildQuery() method, the value of each link node
// will pass to the current operator.  Once the query is built,
// the input node value will be reset to the original value
public void setInputNodeValue(int nNode, String value) {
  // empty body - should extended by each child class
  switch (nNode) {
    case 1:          // relation 1
      setRelation2(value);
      break;
    case 2:          // relation 2
      setRelation2(value);
      break;
    case 3:          // conditon
      setCondition(value);
      break;
    default:
  }
}
// mouse events on this class
public void mouseClicked(MouseEvent e) {
  // call the base class event
  super.mouseClicked(e);

  // if this is right mouse clicked, then popup
  // property window
  if (!isInBound(e.getX(), e.getY())) {
    return ;
  }

  // Check if this operator should be redrawn
  // this condition also prevents the property window
  // pop up twice.
  if (isDirty()) {
    return;
  }
```

```
    if ( e.getModifiers() == e.BUTTON3_MASK) {

      for (int i=0; i<labels.length; ++i) {
        String fieldText="";

        // Assigned the filed name
        propertyLabels[i] = new JLabel(labels[i]);
        // Get the filed information
        switch (i) {
          case 0:           // name
            fieldText = new String(getName());
            break;
          case 1:           // position X
            fieldText = String.valueOf(getX());
            break;
          case 2:           // position Y
            fieldText = String.valueOf(getY());
            break;
          case 3:           // relation 1
            fieldText = relation1;
            break;
          case 4:           // relation 2
            fieldText = relation2;
            break;
          case 5:           // condition
            fieldText = condition;
            break;
          default:
        }
        propertyTextFields[i] = new JTextField(new String(fieldText));
      }

      // System.out.println("popup the property window");

      PropertyWindow propertyWindow = new
PropertyWindow(FrameMain.parent,
                                                    propertyLabels,
propertyTextFields);

      if (propertyWindow.propertyOption == propertyWindow.OK) {
        // Save the changes
        for (int i=0; i<labels.length; ++i) {

          String fieldText = propertyTextFields[i].getText();

          // Get the field inforamtion from the text field
          switch (i) {
            case 0:           // name
              setName(fieldText);
              break;
            case 1:           // position X
              setX((new Double(fieldText)).doubleValue());
              break;
            case 2:           // position Y
              setY((new Double(fieldText)).doubleValue());
              break;
```

```
            case 3:             // relation 1
              setRelation1(fieldText);
              break;
            case 4:             // relation 2
              setRelation2(fieldText);
              break;
            case 5:             // condition
              setCondition(fieldText);
              break;
            default:
          }

        }

      }
      propertyWindow.dispose();

      // this operator should be redrawn
      setDirty(true);
    }

  }


  public void draw(Graphics g) {
    if (g == null) {
      System.out.println("graphics is null on OperatorJoin->draw()");
      return;
    }

    // Draw the main body
    super.draw(g);
    // Draw the nodes
    super.drawThreeInputNodes(g);
    // Draw the relation 1 name
    drawRelation1(g);
    // Draw the relation 2 name
    drawRelation2(g);

    // Draw the condition string
    drawCondition(g);

    // now, the operator should not be redrawn again
    super.setDirty(false);

  }

  public void drawRelation1(Graphics graphics) {

    double relationX = 0;
    double relationY = 0;
    String relation = "";

    // reference to relation 1
    relation = relation1;

    if (graphics == null) {
      return;
```

```
    }

    if (relation.length() == 0) {
      return;
    }


    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;

    // Make sure this oper
    if (isInCollection(relation)) {
      // Draw the connection line to output node (ox, oy) on relation
      // System.out.println(relation + " is in the collection");
      Operator op = (Operator) getObject(relation);

      g.drawLine((int) (op.ox + Operator.NODE_RADIUS/2),
                 (int) (op.oy + Operator.NODE_RADIUS),
                 (int) (this.x1 + Operator.NODE_RADIUS/2),
                 (int) (this.y1));

    } else {
      // Draw the operator name and connection line
      //System.out.println(relation + " is not in the collection");

      relationX = x1 - getWidth()/2;
      relationY = getY() - getHeight();


      // draw the node
      Shape relationNode = new Ellipse2D.Double(relationX,
                                                relationY,

Operator.NODE_RADIUS,NODE_RADIUS);

      g.draw(relationNode);

      // Draw the conection between two nodes
      g.drawLine((int) (x1 + Operator.NODE_RADIUS/2),
                 (int) y1,
                 (int) (relationX + Operator.NODE_RADIUS/2),
                 (int) (relationY + Operator.NODE_RADIUS));

      // Calculate the length of the name

      Font holdFont = graphics.getFont();
      graphics.setFont(new Font(holdFont.getName(),
                                holdFont.getStyle(),
                                holdFont.getSize() - 2));
      FontMetrics fm = graphics.getFontMetrics();

      // System.out.println("Font size = " + holdFont.getSize());

      double nameWidth = fm.stringWidth(relation) + 2;
      double nameHeight = fm.getHeight() + 6;

      // draw a straight line
      g.drawLine((int) (relationX - (nameWidth/2)),(int) relationY,
```

```
                (int) (relationX + (nameWidth/2)),(int) relationY);

    // draw the relation name
    g.drawString(relation,
                 (int) (relationX - (nameWidth/2) + 1),
                 (int) (relationY - 3));


   // after all, change the font back to original
   graphics.setFont(holdFont);

  }


}

public void drawRelation2(Graphics graphics) {

  double relationX = 0;
  double relationY = 0;
  String relation = "";

  // reference to relation 2
  relation = relation2;

  if (graphics == null) {
    return;
  }

  if (relation.length() == 0) {
    return;
  }


  // Use the Graphics2D object
  Graphics2D g = (Graphics2D) graphics;

  // Make sure this oper
  if (isInCollection(relation)) {
    // Draw the connection line to output node (ox, oy) on relation
    // System.out.println(relation + " is in the collection");
    Operator op = (Operator) getObject(relation);

    g.drawLine((int) (op.ox + Operator.NODE_RADIUS/2),
               (int) (op.oy + Operator.NODE_RADIUS),
               (int) (this.x2 + Operator.NODE_RADIUS/2),
               (int) (this.y2));

  } else {
    // Draw the operator name and connection line
    //System.out.println(relation + " is not in the collection");

    relationX = x2;
    relationY = y2 - getHeight() * 2;   // twice's high


    // draw the node
    Shape relationNode = new Ellipse2D.Double(relationX,
```

```
                                                    relationY,
Operator.NODE_RADIUS,NODE_RADIUS);

      g.draw(relationNode);

      // Draw the conection between two nodes
      g.drawLine((int) (x2 + Operator.NODE_RADIUS/2),
                 (int) y2,
                 (int) (relationX + Operator.NODE_RADIUS/2),
                 (int) (relationY + Operator.NODE_RADIUS));

      // Calculate the length of the name

      Font holdFont = graphics.getFont();
      graphics.setFont(new Font(holdFont.getName(),
                                holdFont.getStyle(),
                                holdFont.getSize() - 2));
      FontMetrics fm = graphics.getFontMetrics();

      // System.out.println("Font size = " + holdFont.getSize());

      double nameWidth = fm.stringWidth(relation) + 2;
      double nameHeight = fm.getHeight() + 6;

      // draw a straight line
      g.drawLine((int) (relationX - (nameWidth/2)),(int) relationY,
                 (int) (relationX + (nameWidth/2)),(int) relationY);

      // draw the relation name
      g.drawString(relation,
                   (int) (relationX - (nameWidth/2) + 1),
                   (int) (relationY - 3));


     // after all, change the font back to original
     graphics.setFont(holdFont);

    }


  }
  public void drawCondition(Graphics graphics) {

    double conditionX = 0;
    double conditionY = 0;

    if (graphics == null) {
      return;
    }

    if (condition.length() == 0) {
      return;
    }

    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;
```

```
    // Draw the condition name and connection line
    conditionX = x3 + getWidth()/2;
    conditionY = getY() - getHeight();


    // draw the node
    Shape conditionNode = new Ellipse2D.Double(conditionX,
                                               conditionY,

Operator.NODE_RADIUS,NODE_RADIUS);

    g.draw(conditionNode);

    // Draw the conection between two nodes
    g.drawLine((int) (x3 + Operator.NODE_RADIUS/2),
               (int) y3,
               (int) (conditionX + Operator.NODE_RADIUS/2),
               (int) (conditionY + Operator.NODE_RADIUS));

    // Calculate the length of the name

    Font holdFont = graphics.getFont();
    graphics.setFont(new Font(holdFont.getName(),
                              holdFont.getStyle(),
                              holdFont.getSize() - 2));
    FontMetrics fm = graphics.getFontMetrics();

    // System.out.println("Font size = " + holdFont.getSize());

    double nameWidth = fm.stringWidth(condition) + 2;
    double nameHeight = fm.getHeight() + 6;

    // draw a straight line
    g.drawLine((int) (conditionX - (nameWidth/2)),(int) conditionY,
               (int) (conditionX + (nameWidth/2)),(int) conditionY);

    // draw the relation name
    g.drawString(condition,
                 (int) (conditionX - (nameWidth/2) + 1),
                 (int) (conditionY - 3));


   // after all, change the font back to original
   graphics.setFont(holdFont);

  }

  // Action area
  public String buildQuery() {
    StringBuffer qry = new StringBuffer("select distinct * from ");
    // Check if the relation is operator object in the collection
    if (isInCollection(relation1)) {
      qry.append(" (" );
      Operator op = (Operator) getObject(relation1);
      qry.append(op.buildQuery());
      qry.append(") r1");
    } else {
      qry.append(relation1).append(" r1");
```

```
    }
    qry.append(", ");
    if (isInCollection(relation2)) {
      qry.append(" (" );
      Operator op = (Operator) getObject(relation2);
      qry.append(op.buildQuery());
      qry.append(") r2");
    } else {
      qry.append(relation2).append(" r2");
    }

    if (condition.length()>0) {
      qry.append(" where ").append(condition);
    }

    return qry.toString();
  }


  // Implement Externalizable interface - write
  public void writeExternal(ObjectOutput out) throws IOException {
    // Call the super class to save the common data
    super.writeExternal(out);

    // Write the data belong to this operator
    out.writeObject(relation1);
    out.writeObject(relation2);
    out.writeObject(condition);

  }


  // Implement Externalizable interface - read
  public void readExternal(ObjectInput in) {
    // call the super class to get the common data
    super.readExternal(in);

    // read the data belongs to this operator
    try {
      relation1 = (String) in.readObject();
      relation2 = (String) in.readObject();
      condition = (String) in.readObject();
   } catch (Exception e) {
      System.err.println(e);
   }
  }
}
```

21. OperatorProject.java

```
/*
 * Author: Ron Chen
 * File: OperatorProject.java
 *
 * A Class for project operator for DFQL
```

```
 *
*/

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import java.io.*;
import java.util.*;


public class OperatorProject extends Operator
{

  final static String OPERATOR_TYPE = "project";

  String[] labels = {"Name", "Position X", "Position Y",
                     "Relation", "Attribute List"};
  JLabel[] propertyLabels = new JLabel[labels.length];
  JTextField[] propertyTextFields = new JTextField[labels.length];

  // ------ Property -----------------
  private String relation="";
  private String list="";

  public OperatorProject() {
    super(OPERATOR_TYPE);
  }

  public OperatorProject(Vector vRef) {
    super(OPERATOR_TYPE, vRef);
  }

  // -------- Property get/set ------------
  public String getRelation() {
    return relation;
  }

  public void setRelation(String sRelation) {
    relation = new String(sRelation);
  }

  public String getList() {
    return list;
  }

  public void setList(String sList) {
    if (!(sList == null)) {
      list = new String(sList);
    }
  }
  // --------- end of property get/set -------

  // This method is specific used by OperatorUsre.java class
  // when the input node node link to the InputBar Node
  // when the buildQuery() method, the value of each link node
  // will pass to the current operator.  Once the query is built,
  // the input node value will be reset to the original value
```

```
public void setInputNodeValue(int nNode, String value) {
  // empty body - should extended by each child class
  switch (nNode) {
    case 1:          // relation
      setRelation(value);
      break;
    case 2:          // condition
      setList(value);
      break;
    default:
  }
}

// mouse events on this class
public void mouseClicked(MouseEvent e) {
  // call the base class event
  super.mouseClicked(e);

  // if this is right mouse clicked, then popup
  // property window
  if (!isInBound(e.getX(), e.getY())) {
    return ;
  }

  // Check if this operator should be redrawn
  // this condition also prevents the property window
  // pop up twice.
  if (isDirty()) {
    return;
  }

  if ( e.getModifiers() == e.BUTTON3_MASK) {


    for (int i=0; i<labels.length; ++i) {
      String fieldText="";

      // Assigned the filed name
      propertyLabels[i] = new JLabel(labels[i]);
      // Get the filed information
      switch (i) {
        case 0:            // name
          fieldText = new String(getName());
          break;
        case 1:            // position X
          fieldText = String.valueOf(getX());
          break;
        case 2:            // position Y
          fieldText = String.valueOf(getY());
          break;
        case 3:            // relation
          fieldText = relation;
          break;
        case 4:            // list
          fieldText = list;
          break;
        default:
      }
```

```
        propertyTextFields[i] = new JTextField(new String(fieldText));
      }

      // System.out.println("popup the property window");

      PropertyWindow propertyWindow = new
PropertyWindow(FrameMain.parent,
                                                    propertyLabels,

propertyTextFields);

      if (propertyWindow.propertyOption == propertyWindow.OK) {
        // Save the changes
        for (int i=0; i<labels.length; ++i) {

          String fieldText = propertyTextFields[i].getText();

          // Get the field inforamtion from the text field
          switch (i) {
            case 0:           // name
              setName(fieldText);
              break;
            case 1:           // position X
              setX((new Double(fieldText)).doubleValue());
              break;
            case 2:           // position Y
              setY((new Double(fieldText)).doubleValue());
              break;
            case 3:           // relation
              setRelation(fieldText);
              break;
            case 4:           // list
              setList(fieldText);
              break;
            default:
          }

        }

      }
      propertyWindow.dispose();

      // this operator should be redrawn
      setDirty(true);
    }

  }

  public void draw(Graphics g) {
    if (g == null) {
      System.out.println("graphics is null on OperatorProject->draw()");
      return;
    }

    // Draw the main body
    super.draw(g);
    // Draw the nodes
    super.drawTwoInputNodes(g);
```

```
      // Draw the relation name
      drawRelation(g);
      // Draw the list string
      drawList(g);

      // now, the operator should not be redrawn again
      super.setDirty(false);

    }

    public void drawRelation(Graphics graphics) {

      double relationX = 0;
      double relationY = 0;

      if (graphics == null) {
        return;
      }

      if (relation.length() == 0) {
        return;
      }

      // Use the Graphics2D object
      Graphics2D g = (Graphics2D) graphics;

      // Make sure this oper
      if (isInCollection(relation)) {
        // Draw the connection line to output node (ox, ox) on relation
        // System.out.println(relation + " is in the collection");
        Operator op = (Operator) getObject(relation);

        g.drawLine((int) (op.ox + Operator.NODE_RADIUS/2),
                   (int) (op.oy + Operator.NODE_RADIUS),
                   (int) (this.x1 + Operator.NODE_RADIUS/2),
                   (int) (this.y1));

      } else {
        // Draw the operator name and connection line
        //System.out.println(relation + " is not in the collection");

        relationX = getX();
        relationY = getY() - getHeight();


        // draw the node
        Shape relationNode = new Ellipse2D.Double(relationX,
                                                  relationY,
Operator.NODE_RADIUS,NODE_RADIUS);

        g.draw(relationNode);

        // Draw the conection between two nodes
        g.drawLine((int) (x1 + Operator.NODE_RADIUS/2),
                   (int) y1,
                   (int) (relationX + Operator.NODE_RADIUS/2),
                   (int) (relationY + Operator.NODE_RADIUS));
```

```
      // Calculate the length of the name

      Font holdFont = graphics.getFont();
      graphics.setFont(new Font(holdFont.getName(),
                                holdFont.getStyle(),
                                holdFont.getSize() - 2));
      FontMetrics fm = graphics.getFontMetrics();

      // System.out.println("Font size = " + holdFont.getSize());

      double nameWidth = fm.stringWidth(relation) + 2;
      double nameHeight = fm.getHeight() + 6;

      // draw a straight line
      g.drawLine((int) (relationX - (nameWidth/2)),(int) relationY,
                 (int) (relationX + (nameWidth/2)),(int) relationY);

      // draw the relation name
      g.drawString(relation,
                   (int) (relationX - (nameWidth/2) + 1),
                   (int) (relationY - 3));


     // after all, change the font back to original
     graphics.setFont(holdFont);

    }


  }

  public void drawList(Graphics graphics) {

    double listX = 0;
    double listY = 0;

    if (graphics == null) {
      return;
    }

    if (list.length() == 0) {
      return;
    }

    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;

    // Draw the list name and connection line
    listX = getX() + getWidth();
    listY = getY() - getHeight();


    // draw the node
    Shape listNode = new Ellipse2D.Double(listX,
                                          listY,

Operator.NODE_RADIUS,NODE_RADIUS);
```

```
    g.draw(listNode);

    // Draw the conection between two nodes
    g.drawLine((int) (x2 + Operator.NODE_RADIUS/2),
               (int) y2,
               (int) (listX + Operator.NODE_RADIUS/2),
               (int) (listY + Operator.NODE_RADIUS));

    // Calculate the length of the name

    Font holdFont = graphics.getFont();
    graphics.setFont(new Font(holdFont.getName(),
                              holdFont.getStyle(),
                              holdFont.getSize() - 2));
    FontMetrics fm = graphics.getFontMetrics();

    // System.out.println("Font size = " + holdFont.getSize());

    double nameWidth = fm.stringWidth(list) + 2;
    double nameHeight = fm.getHeight() + 6;

    // draw a straight line
    g.drawLine((int) (listX - (nameWidth/2)),(int) listY,
               (int) (listX + (nameWidth/2)),(int) listY);

    // draw the relation name
    g.drawString(list,
                 (int) (listX - (nameWidth/2) + 1),
                 (int) (listY - 3));


   // after all, change the font back to original
   graphics.setFont(holdFont);

  }

  // Action area
  public String buildQuery() {
    StringBuffer qry = new StringBuffer("select distinct ");
    // Obtain the attribute list
    if (list.length() > 0 ) {
      qry.append(list);
    } else {
      // if no specify, assume all attributes
      qry.append("*");
    }
    qry.append(" from ");
    // Check if the relation is operator object in the collection
    if (isInCollection(relation)) {
      qry.append(" (" );
      Operator op = (Operator) getObject(relation);
      qry.append(op.buildQuery());
      qry.append(")");
    } else {
      qry.append(relation);
    }
```

```
    return qry.toString();
  }


  // Implement Externalizable interface - write
  public void writeExternal(ObjectOutput out) throws IOException {
    // Call the super class to save the common data
    super.writeExternal(out);

    // Write the data belong to this operator
    out.writeObject(relation);
    out.writeObject(list);

  }


  // Implement Externalizable interface - read
  public void readExternal(ObjectInput in) {
    // call the super class to get the common data
    super.readExternal(in);

    // read the data belongs to this operator
    try {
      relation = (String) in.readObject();
      list = (String) in.readObject();
   } catch (Exception e) {
      System.err.println(e);
   }
  }

}
```

22. OperatorSelect.java

```
/*
 * File: OperatorSelect.java
 * Author: Ron Chen
 *
 * A Class for select operator for DFQL
 *
*/

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import java.io.*;
import java.util.*;


public class OperatorSelect extends Operator
{

  final static String OPERATOR_TYPE = "select";
```

```
String[] labels = {"Name", "Position X", "Position Y",
                   "Relation", "Condition"};
JLabel[] propertyLabels = new JLabel[labels.length];
JTextField[] propertyTextFields = new JTextField[labels.length];

// ------ Property -----------------
private String relation="";
private String condition="";

// ------ end of property ----------

public OperatorSelect() {
  super(OPERATOR_TYPE);
}

public OperatorSelect(Vector vRef) {
  super(OPERATOR_TYPE, vRef);
}

// -------- Property get/set ------------
public String getRelation() {
  return relation;
}

public void setRelation(String sRelation) {
  relation = new String(sRelation);
}

public String getCondition() {
  return condition;
}

public void setCondition(String sCondition) {
  if (!(sCondition == null)) {
    condition = new String(sCondition);
  }
}
// --------- end of property get/set -------

// This method is specific used by OperatorUsre.java class
// when the input node node link to the InputBar Node
// when the buildQuery() method, the value of each link node
// will pass to the current operator.  Once the query is built,
// the input node value will be reset to the original value
public void setInputNodeValue(int nNode, String value) {
  // empty body - should extended by each child class
  switch (nNode) {
    case 1:          // relation
      setRelation(value);
      break;
    case 2:          // condition
      setCondition(value);
      break;
    default:
  }
}
```

```
  // mouse events on this class
  public void mouseClicked(MouseEvent e) {
    // call the base class event
    super.mouseClicked(e);

    // if this is right mouse clicked, then popup
    // property window
    if (!isInBound(e.getX(), e.getY())) {
      return ;
    }

    // Check if this operator should be redrawn
    // this condition also prevents the property window
    // pop up twice.
    if (isDirty()) {
      return;
    }

    if ( e.getModifiers() == e.BUTTON3_MASK) {
      for (int i=0; i<labels.length; ++i) {
        String fieldText="";

        // Assigned the filed name
        propertyLabels[i] = new JLabel(labels[i]);
        // Get the filed information
        switch (i) {
          case 0:           // name
            fieldText = new String(getName());
            break;
          case 1:           // position X
            fieldText = String.valueOf(getX());
            break;
          case 2:           // position Y
            fieldText = String.valueOf(getY());
            break;
          case 3:           // relation
            fieldText = relation;
            break;
          case 4:           // condition
            fieldText = condition;
            break;
          default:
        }
        propertyTextFields[i] = new JTextField(new String(fieldText));
      }

      // System.out.println("popup the property window");

      PropertyWindow propertyWindow = new
PropertyWindow(FrameMain.parent,
                                                    propertyLabels,

propertyTextFields);

      if (propertyWindow.propertyOption == propertyWindow.OK) {
        // Save the changes
        for (int i=0; i<labels.length; ++i) {
```

```
            String fieldText = propertyTextFields[i].getText();

            // Get the field inforamtion from the text field
            switch (i) {
              case 0:           // name
                setName(fieldText);
                break;
              case 1:           // position X
                setX((new Double(fieldText)).doubleValue());
                break;
              case 2:           // position Y
                setY((new Double(fieldText)).doubleValue());
                break;
              case 3:           // relation
                setRelation(fieldText);
                break;
              case 4:           // condition
                setCondition(fieldText);
                break;
              default:
            }

          }

        }
        propertyWindow.dispose();

        // this operator should be redrawn
        setDirty(true);
      }

    }

    public void draw(Graphics g) {
      if (g == null) {
        System.out.println("graphics is null on OperatorSelect->draw()");
        return;
      }

      // Draw the main body
      super.draw(g);
      // Draw the nodes
      super.drawTwoInputNodes(g);
      // Draw the relation name
      drawRelation(g);
      // Draw the condition string
      drawCondition(g);

      // now, the operator should not be redrawn again
      super.setDirty(false);

    }

    public void drawRelation(Graphics graphics) {

      double relationX = 0;
      double relationY = 0;
```

```
    if (graphics == null) {
      return;
    }

    if (relation.length() == 0) {
      return;
    }

    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;

    // Check if this operator is in the collection
    if (isInCollection(relation)) {
      // Draw the connection line to output node (ox, oy) on relation
      // System.out.println(relation + " is in the collection");
      Operator op = (Operator) getObject(relation);

      g.drawLine((int) (op.ox + Operator.NODE_RADIUS/2),
                 (int) (op.oy + Operator.NODE_RADIUS),
                 (int) (this.x1 + Operator.NODE_RADIUS/2),
                 (int) (this.y1));

    } else {
      // Draw the operator name and connection line
      //System.out.println(relation + " is not in the collection");

      relationX = getX();
      relationY = getY() - getHeight();


      // draw the node
      Shape relationNode = new Ellipse2D.Double(relationX,
                                                relationY,

Operator.NODE_RADIUS,NODE_RADIUS);

      g.draw(relationNode);

      // Draw the conection between two nodes
      g.drawLine((int) (x1 + Operator.NODE_RADIUS/2),
                 (int) y1,
                 (int) (relationX + Operator.NODE_RADIUS/2),
                 (int) (relationY + Operator.NODE_RADIUS));

      // Calculate the length of the name

      Font holdFont = graphics.getFont();
      graphics.setFont(new Font(holdFont.getName(),
                                holdFont.getStyle(),
                                holdFont.getSize() - 2));
      FontMetrics fm = graphics.getFontMetrics();

      // System.out.println("Font size = " + holdFont.getSize());

      double nameWidth = fm.stringWidth(relation) + 2;
      double nameHeight = fm.getHeight() + 6;

      // draw a straight line
```

```
        g.drawLine((int) (relationX - (nameWidth/2)),(int) relationY,
                   (int) (relationX + (nameWidth/2)),(int) relationY);

        // draw the relation name
        g.drawString(relation,
                     (int) (relationX - (nameWidth/2) + 1),
                     (int) (relationY - 3));


       // after all, change the font back to original
       graphics.setFont(holdFont);

      }


    }

    public void drawCondition(Graphics graphics) {

      double conditionX = 0;
      double conditionY = 0;

      if (graphics == null) {
        return;
      }

      if (condition.length() == 0) {
        return;
      }

      // Use the Graphics2D object
      Graphics2D g = (Graphics2D) graphics;

      // Draw the condition name and connection line
      conditionX = getX() + getWidth();
      conditionY = getY() - getHeight();


      // draw the node
      Shape conditionNode = new Ellipse2D.Double(conditionX,
                                                 conditionY,
Operator.NODE_RADIUS,NODE_RADIUS);

      g.draw(conditionNode);

      // Draw the conection between two nodes
      g.drawLine((int) (x2 + Operator.NODE_RADIUS/2),
                 (int) y2,
                 (int) (conditionX + Operator.NODE_RADIUS/2),
                 (int) (conditionY + Operator.NODE_RADIUS));

      // Calculate the length of the name

      Font holdFont = graphics.getFont();
      graphics.setFont(new Font(holdFont.getName(),
                                holdFont.getStyle(),
                                holdFont.getSize() - 2));
```

```
  FontMetrics fm = graphics.getFontMetrics();

  // System.out.println("Font size = " + holdFont.getSize());

  double nameWidth = fm.stringWidth(condition) + 2;
  double nameHeight = fm.getHeight() + 6;

  // draw a straight line
  g.drawLine((int) (conditionX - (nameWidth/2)),(int) conditionY,
             (int) (conditionX + (nameWidth/2)),(int) conditionY);

  // draw the relation name
  g.drawString(condition,
               (int) (conditionX - (nameWidth/2) + 1),
               (int) (conditionY - 3));


 // after all, change the font back to original
 graphics.setFont(holdFont);

}

// Action area
public String buildQuery() {
  StringBuffer qry = new StringBuffer("select distinct * from ");
  // Check if the relation is operator object in the collection
  if (isInCollection(relation)) {
    qry.append(" (" );
    Operator op = (Operator) getObject(relation);
    qry.append(op.buildQuery());
    qry.append(")");
  } else {
    qry.append(relation);
  }
  if (condition.length()>0) {
    qry.append(" where ").append(condition);
  }

  return qry.toString();
}

// Implement Externalizable interface - write
public void writeExternal(ObjectOutput out) throws IOException {
  // Call the super class to save the common data
  super.writeExternal(out);

  // Write the data belong to this operator
  out.writeObject(relation);
  out.writeObject(condition);

}


// Implement Externalizable interface - read
public void readExternal(ObjectInput in) {
  // call the super class to get the common data
  super.readExternal(in);
```

```
    // read the data belongs to this operator
    try {
      relation = (String) in.readObject();
      condition = (String) in.readObject();
      // System.out.println("relation = " + relation + " condition = " +
condition);
   } catch (Exception e) {
      System.err.println(e);
   }
  }
}
```

23. OperatorUnion.java

```
/*
 * Author: Ron Chen
 * File: OperatorUnion.java
 *
 * A Class for union operator for DFQL
 *
*/

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import java.io.*;
import java.util.*;


public class OperatorUnion extends Operator
{

  final static String OPERATOR_TYPE = "union";

  String[] labels = {"Name", "Position X", "Position Y",
                     "Relation 1", "Relation 2"};
  JLabel[] propertyLabels = new JLabel[labels.length];
  JTextField[] propertyTextFields = new JTextField[labels.length];

  // ------ Property -----------------
  private String relation1="";
  private String relation2="";

  // ------ end of property ----------

  public OperatorUnion() {
    super(OPERATOR_TYPE);
  }

  public OperatorUnion(Vector vRef) {
    super(OPERATOR_TYPE, vRef);
  }

  // -------- Property get/set ------------
```

```
public String getRelation1() {
  return relation1;
}

public void setRelation1(String sRelation) {
  relation1 = new String(sRelation);
}

public String getRelation2() {
  return relation2;
}

public void setRelation2(String sRelation) {
  relation2 = new String(sRelation);
}

// --------- end of property get/set -------

// This method is specific used by OperatorUsre.java class
// when the input node node link to the InputBar Node
// when the buildQuery() method, the value of each link node
// will pass to the current operator.  Once the query is built,
// the input node value will be reset to the original value
public void setInputNodeValue(int nNode, String value) {
  // empty body - should extended by each child class
  switch (nNode) {
    case 1:          // relation 1
      setRelation1(value);
      break;
    case 2:          // relation 2
      setRelation2(value);
      break;
    default:
  }
}

// mouse events on this class
public void mouseClicked(MouseEvent e) {
  // call the base class event
  super.mouseClicked(e);

  // if this is right mouse clicked, then popup
  // property window
  if (!isInBound(e.getX(), e.getY())) {
    return ;
  }

  // Check if this operator should be redrawn
  // this condition also prevents the property window
  // pop up twice.
  if (isDirty()) {
    return;
  }

  if ( e.getModifiers() == e.BUTTON3_MASK) {
    for (int i=0; i<labels.length; ++i) {
      String fieldText="";
```

```
        // Assigned the filed name
        propertyLabels[i] = new JLabel(labels[i]);
        // Get the filed information
        switch (i) {
          case 0:           // name
            fieldText = new String(getName());
            break;
          case 1:           // position X
            fieldText = String.valueOf(getX());
            break;
          case 2:           // position Y
            fieldText = String.valueOf(getY());
            break;
          case 3:           // relation 1
            fieldText = relation1;
            break;
          case 4:           // relation 2
            fieldText = relation2;
            break;
          default:
        }
        propertyTextFields[i] = new JTextField(new String(fieldText));
      }

      // System.out.println("popup the property window");

      PropertyWindow propertyWindow = new
PropertyWindow(FrameMain.parent,
                                                    propertyLabels,

propertyTextFields);

      if (propertyWindow.propertyOption == propertyWindow.OK) {
        // Save the changes
        for (int i=0; i<labels.length; ++i) {

          String fieldText = propertyTextFields[i].getText();

          // Get the field inforamtion from the text field
          switch (i) {
            case 0:         // name
              setName(fieldText);
              break;
            case 1:           // position X
              setX((new Double(fieldText)).doubleValue());
              break;
            case 2:           // position Y
              setY((new Double(fieldText)).doubleValue());
              break;
            case 3:           // relation 1
              setRelation1(fieldText);
              break;
            case 4:           // relation 2
              setRelation2(fieldText);
              break;
            default:
          }
```

```
        }

      }
      propertyWindow.dispose();

      // this operator should be redrawn
      setDirty(true);
    }

  }

  public void draw(Graphics g) {
    if (g == null) {
      System.out.println("graphics is null on OperatorUnion->draw()");
      return;
    }

    // Draw the main body
    super.draw(g);
    // Draw the nodes
    super.drawTwoInputNodes(g);

    // Draw the relation 1 name
    drawRelation1(g);
    // Draw the relation 2 name
    drawRelation2(g);

    // now, the operator should not be redrawn again
    super.setDirty(false);

  }

  public void drawRelation1(Graphics graphics) {

    double relationX = 0;
    double relationY = 0;
    String relation = "";

    relation = relation1;

    if (graphics == null) {
      return;
    }

    if (relation.length() == 0) {
      return;
    }

    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;

    // Make sure this oper
    if (isInCollection(relation)) {
      // Draw the connection line to output node (ox, oy) on relation
      // System.out.println(relation + " is in the collection");
      Operator op = (Operator) getObject(relation);

      g.drawLine((int) (op.ox + Operator.NODE_RADIUS/2),
```

```
                    (int) (op.oy + Operator.NODE_RADIUS),
                    (int) (this.x1 + Operator.NODE_RADIUS/2),
                    (int) (this.y1));

      } else {
        // Draw the operator name and connection line
        //System.out.println(relation + " is not in the collection");

        relationX = getX();
        relationY = getY() - getHeight();


        // draw the node
        Shape relationNode = new Ellipse2D.Double(relationX,
                                                  relationY,

Operator.NODE_RADIUS,NODE_RADIUS);

        g.draw(relationNode);

        // Draw the conection between two nodes
        g.drawLine((int) (x1 + Operator.NODE_RADIUS/2),
                   (int) y1,
                   (int) (relationX + Operator.NODE_RADIUS/2),
                   (int) (relationY + Operator.NODE_RADIUS));

        // Calculate the length of the name

        Font holdFont = graphics.getFont();
        graphics.setFont(new Font(holdFont.getName(),
                                  holdFont.getStyle(),
                                  holdFont.getSize() - 2));
        FontMetrics fm = graphics.getFontMetrics();

        // System.out.println("Font size = " + holdFont.getSize());

        double nameWidth = fm.stringWidth(relation) + 2;
        double nameHeight = fm.getHeight() + 6;

        // draw a straight line
        g.drawLine((int) (relationX - (nameWidth/2)),(int) relationY,
                   (int) (relationX + (nameWidth/2)),(int) relationY);

        // draw the relation name
        g.drawString(relation,
                     (int) (relationX - (nameWidth/2) + 1),
                     (int) (relationY - 3));


       // after all, change the font back to original
       graphics.setFont(holdFont);

      }


    }

    public void drawRelation2(Graphics graphics) {
```

```
    double relationX = 0;
    double relationY = 0;

    String relation = "";

    relation = relation2;

    if (graphics == null) {
      return;
    }

    if (relation.length() == 0) {
      return;
    }

    // Use the Graphics2D object
    Graphics2D g = (Graphics2D) graphics;

    // Make sure this oper
    if (isInCollection(relation)) {
      // Draw the connection line to output node (ox, oy) on relation
      // System.out.println(relation + " is in the collection");
      Operator op = (Operator) getObject(relation);

      g.drawLine((int) (op.ox + Operator.NODE_RADIUS/2),
                 (int) (op.oy + Operator.NODE_RADIUS),
                 (int) (this.x2 + Operator.NODE_RADIUS/2),
                 (int) (this.y2));

    } else {
      // Draw the operator name and connection line
      //System.out.println(relation + " is not in the collection");

      relationX = getX() + getWidth();
      relationY = getY() - getHeight();


      // draw the node
      Shape relationNode = new Ellipse2D.Double(relationX,
                                                relationY,

Operator.NODE_RADIUS,NODE_RADIUS);

      g.draw(relationNode);

      // Draw the conection between two nodes
      g.drawLine((int) (x2 + Operator.NODE_RADIUS/2),
                 (int) y2,
                 (int) (relationX + Operator.NODE_RADIUS/2),
                 (int) (relationY + Operator.NODE_RADIUS));

      // Calculate the length of the name

      Font holdFont = graphics.getFont();
      graphics.setFont(new Font(holdFont.getName(),
                                holdFont.getStyle(),
                                holdFont.getSize() - 2));
```

```
    FontMetrics fm = graphics.getFontMetrics();

    // System.out.println("Font size = " + holdFont.getSize());

    double nameWidth = fm.stringWidth(relation) + 2;
    double nameHeight = fm.getHeight() + 6;

    // draw a straight line
    g.drawLine((int) (relationX - (nameWidth/2)),(int) relationY,
               (int) (relationX + (nameWidth/2)),(int) relationY);

    // draw the relation name
    g.drawString(relation,
                 (int) (relationX - (nameWidth/2) + 1),
                 (int) (relationY - 3));


   // after all, change the font back to original
   graphics.setFont(holdFont);

  }
}

// Action area
public String buildQuery() {
  StringBuffer qry = new StringBuffer("select distinct * from ");
  // Check if the relation is operator object in the collection
  if (isInCollection(relation1)) {
    qry.append(" (" );
    Operator op = (Operator) getObject(relation1);
    qry.append(op.buildQuery());
    qry.append(")");
  } else {
    qry.append(relation1);
  }
  qry.append(" union ");
  qry.append("select distinct * from ");
  if (isInCollection(relation2)) {
    qry.append(" (" );
    Operator op = (Operator) getObject(relation2);
    qry.append(op.buildQuery());
    qry.append(")");
  } else {
    qry.append(relation2);
  }

  return qry.toString();
}


// Implement Externalizable interface - write
public void writeExternal(ObjectOutput out) throws IOException {
  // Call the super class to save the common data
  super.writeExternal(out);

  // Write the data belong to this operator
  out.writeObject(relation1);
  out.writeObject(relation2);
```

```
  }

  // Implement Externalizable interface - read
  public void readExternal(ObjectInput in) {
    // call the super class to get the common data
    super.readExternal(in);

    // read the data belongs to this operator
    try {
      relation1 = (String) in.readObject();
      relation2 = (String) in.readObject();
   } catch (Exception e) {
      System.err.println(e);
   }
  }
}
```

24. OperatorUser.java

```
/*
 * Author: Ron Chen
 * File: OperatorUser.java
 *
 * A Class that extends the DFQL Operator class, but
 * uses for user defined operator
 *
 * This is a very complex and long class, need to pay a lot of
 * attention during the coding.
 *
 * There are two major situations that must be paid
 * close attention
 * 1. DESIGN mode - user wants to define a DFQL operator
 * 2. INUSED mode - user uses the current user defined DFQL operator
 *
*/

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
import java.io.*;
import java.util.*;


public class OperatorUser extends Operator
{

  final static String OPERATOR_TYPE = "user";

  public Vector vRefUserOperator = null;

  /* DESIGN property */
```

```
  String[] designLabels = {"Name", "Number of Relations", "Number of
Conditions",
                           "Number of Attributes"};
  JLabel[] designPropertyLabels = new JLabel[designLabels.length];
  JTextField[] designPropertyTextFields = new
JTextField[designLabels.length];


  // ---- Property of this class on design mode

  // user operator name
  String designName = "";
  // number of relations
  int numRelations = 0;
  // number of conditions
  int numConditions = 0;
  // number of Attributes
  int numAttributes = 0;

  // ---- end of property

  // default input bar information
  double inputBarX = 80;
  double inputBarY = 80;
  double inputBarWidth = 200;
  double inputBarHeight = 20;

  // Determine the use mode of the operator
  // Design or use
  private int mode;

  // totalNodes
  int totalNodes = numRelations + numConditions + numAttributes;

  // Hold the design canvas
  private JComponent designCanvas = null;

  public OperatorUser() {
   // empty body
  }


  public String getDesignName() {
    return (new String(designName));
  }

  public int getMode() {
    return mode;
  }

  public void setMode(int nMode) {
    mode = nMode;
  }

  // This mothod must be called at the initial creation
  public void setDesignCanvas(JComponent c) {
    designCanvas = c;
  }
```

```
/* --------- Mouse events -------------------- */
public void mouseDragged(MouseEvent e) {
  if (mode == DFQL.INUSED) {
    super.mouseDragged(e);
  }
}

public void mouseMoved(MouseEvent e) {
  if (mode == DFQL.INUSED) {
    super.mouseMoved(e);
  }
}

public void mousePressed(MouseEvent e) {
  if (mode == DFQL.INUSED) {
    super.mousePressed(e);
  }
}

public void mouseEntered(MouseEvent e) {
  if (mode == DFQL.INUSED) {
    super.mouseExited(e);
  }
}

public void mouseExited(MouseEvent e) {
  if (mode == DFQL.INUSED) {
    super.mouseExited(e);
  }

}

public void mouseClicked(MouseEvent e) {
  InputBarNode iNode = null;

  super.mouseClicked(e);

  // This is INUSED mode
  if (mode == DFQL.INUSED) {
    if (!isInBound(e.getX(), e.getY())) {
      return ;
    }

    // Check if this operator should be redrawn
    // this condition also prevents the property window
    // pop up twice.
    if (isDirty()) {
      return;
    }

    if ( e.getModifiers() == e.BUTTON3_MASK) {
      // Computer total nodes
      totalNodes = numRelations + numConditions + numAttributes;

      if ((totalNodes < 1) || (totalNodes >4)) {
        JOptionPane.showMessageDialog(null,
                                      "Too a few or too much nodes",
```

```
                                        "Future feature",
                                        JOptionPane.ERROR_MESSAGE);
          return ;

        }

        int total = (numRelations + numConditions);
        // first 3 labels: Name, Position X, Position Y
        //System.out.println("Create array of lables");
        String[] labels = new String[3 + totalNodes];
        //System.out.println("now, label.length = " + labels.length);
        // Set up the label information
        for (int i=0; i<(labels.length); i++) {
          if (i==0) {
            labels[i] = new String("Name");
          }

          if (i==1) {
            labels[i] = new String("Position X");
          }

           if (i==2) {
            labels[i] = new String("Position Y");
          }

          if ((i > 2) && (i <= (2+numRelations))) {
            labels[i] = new String("Relation " + (i-2));
          }
          // condition node
          if ((numConditions > 0) && (i>(2+numRelations)) &&
(i<=(2+total))) {
            labels[i] = new String("Condition " + (i - 2 -
numRelations));
          }
          // attributes node
          if ((numAttributes > 0) && (i>(2+total))) {
            labels[i] = new String("Attribute " + (i - 2 - total));
          }

        }
        JLabel[] propertyLabels = new JLabel[labels.length];
        JTextField[] propertyTextFields = new JTextField[labels.length];


        for (int i=0; i<(labels.length); ++i) {
          String fieldText="";

          if (i>=3) {
            iNode = (InputBarNode) getNode(i-2);
          }
          // Assigned the filed name
          propertyLabels[i] = new JLabel(labels[i]);

          // Get the filed information
          switch (i) {
            case 0:           // name
              fieldText = new String(getName());
              break;
```

```
            case 1:             // position X
              fieldText = String.valueOf(getX());
              break;
            case 2:             // position Y
              fieldText = String.valueOf(getY());
              break;
            default:
              fieldText = iNode.getInputValue();
              break;
          }
           propertyTextFields[i] = new JTextField(new String(fieldText));
        }

        // System.out.println("popup the property window");

        PropertyWindow propertyWindow = new
PropertyWindow(FrameMain.parent,

propertyLabels,

propertyTextFields);

        if (propertyWindow.propertyOption == propertyWindow.OK) {
          // Save the changes
          for (int i=0; i<(labels.length); ++i) {

            String fieldText = propertyTextFields[i].getText();

            if (i>=3) {
              iNode = (InputBarNode) getNode(i-2);
            }

            // Get the field inforamtion from the text field
            switch (i) {
              case 0:             // name
                setName(fieldText);
                break;
              case 1:             // position X
                setX((new Double(fieldText)).doubleValue());
                break;
              case 2:             // position Y
                setY((new Double(fieldText)).doubleValue());
                break;
              default:
                iNode.setInputValue(new String(fieldText));

            }

          }

        }
        propertyWindow.dispose();

        // this operator should be redrawn
        setDirty(true);
      }
```

```
      // Exit the INUSED mode
      return;
    }


    // This is design mode

    // Check if this operator should be redrawn
    // this condition also prevents the property window
    // pop up twice.
    if (isDirty()) {
      return;
    }


    // On design mode
    if ( e.getModifiers() == e.BUTTON3_MASK) {
      int pickNode = whichNode(e.getX(), e.getY());

      // Check if there is a node
      if (pickNode == 0) {
        return;
      }

      //System.out.println("OperatorUser->mouseClicked(), pickNode = " +
pickNode);

      iNode = (InputBarNode) getNode(pickNode);
      iNode.setProperty();

      setDirty(true);

    }

  }

  /* ---------- End of Mouse Events --------------- */

  // Action Mode

  // User wants to design a new operator
  public void newOperator() {
    if (designCanvas == null) {
      System.out.println("graphics is null on OperatorUser-
>newOperator()");
      return;
    }

    // Ask the information for the new operator design
    askDesign();

    // Draw the input bar
    drawInputBar();

  }


  public void askDesign() {
```

```
    for (int i=0; i<designLabels.length; ++i) {
      String fieldText="";

      // Assigned the filed name
      designPropertyLabels[i] = new JLabel(designLabels[i]);
      // Get the filed information
      switch (i) {
        case 0:           // name
          fieldText = new String(designName);
          break;
        case 1:           // number of relations
          fieldText = String.valueOf(numRelations);
          break;
        case 2:           // number of conditons
          fieldText = String.valueOf(numConditions);
          break;
        case 3:           // number of attributes
          fieldText = String.valueOf(numAttributes);
          break;
        default:
      }
      designPropertyTextFields[i] = new JTextField(new
String(fieldText));
    }

    // System.out.println("popup the property window");

    PropertyWindow propertyWindow = new PropertyWindow(FrameMain.parent,

designPropertyLabels,

designPropertyTextFields);

    if (propertyWindow.propertyOption == propertyWindow.OK) {
      // Save the changes
      for (int i=0; i<designLabels.length; ++i) {

        String fieldText = designPropertyTextFields[i].getText();

        // Get the field inforamtion from the text field
        switch (i) {
          case 0:           // design name
            designName = new String(fieldText);
            break;
          case 1:           // number of relations
            numRelations = (new Integer(fieldText)).intValue();
            break;
          case 2:           // number of conditions
            numConditions = (new Integer(fieldText)).intValue();
            break;
          case 3:           // number of attributes
            numAttributes = (new Integer(fieldText)).intValue();
            break;
          default:
        }

      }
```

```
    }
    propertyWindow.dispose();

    // Build the InputBarNode base on the input
    totalNodes = numRelations + numConditions + numAttributes;
    // no drawing if total nodes is 0
    if (totalNodes == 0) {
      return;
    }

    for (int i=1; i<=totalNodes; i++) {
      InputBarNode iNode = new InputBarNode(i);
      // relation node
      if (i <= numRelations) {
        iNode.setNodeType(InputBarNode.RELATION);
      }
      int total = (numRelations + numConditions);
      // condition node
      if ((numConditions > 0) && (i>numRelations) && (i<=total)) {
        iNode.setNodeType(InputBarNode.CONDITION);
      }
      // attributes node
      if ((numAttributes > 0) && (i>total)) {
        iNode.setNodeType(InputBarNode.ATTRIBUTE);
      }

      // place this node in the collection object
      vRefUserOperator.add(iNode);
    }

  }

  public void drawInputBar() {
    Graphics2D g = (Graphics2D) designCanvas.getGraphics();

    drawInputBar(g);
  }

  public void drawInputBar(Graphics graphics) {
    // System.out.println("OperatorUser->drawInputBar()");

    Graphics2D g = (Graphics2D) graphics;

    if (g == null) {
      System.out.println("graphics is null on OperatorUser-
>drawInputBar()");
      return;
    }

    // no drawing if the designName is not specified
    if (designName.length() == 0) {
      return;
    }

    // Clear up everything on the component canvas
    //System.out.println("OperatorUser->drawInputBar(), call
designCanvas.update(g)");
```

```
    //designCanvas.update((Graphics) g);

    totalNodes = numRelations + numConditions + numAttributes;
    // no drawing if total nodes is 0
    if (totalNodes == 0) {
      return;
    }

    // Hole the current font
    Font holdFont = g.getFont();
    g.setFont(new Font(holdFont.getName(),
                       holdFont.getStyle(),
                       holdFont.getSize() + 2));
    FontMetrics fm = g.getFontMetrics();

    double nameWidth = fm.stringWidth(designName);
    double nameHeight = fm.getHeight();


    double drawAreaWidth = designCanvas.getWidth();

    double designNameX = (drawAreaWidth - nameWidth)/2;
    double designNameY = 20;

    g.drawString(designName, (int) designNameX, (int) designNameY);

    inputBarX = 80;
    inputBarY = nameHeight + designNameY + 40;
    inputBarWidth = drawAreaWidth - inputBarX * 2;
    inputBarHeight = 20;

    g.drawRoundRect((int) inputBarX, (int) inputBarY,
                    (int) inputBarWidth, (int) inputBarHeight,
                          5, 5);
    /*
    g.fillRoundRect((int) inputBarX, (int) inputBarY,
                    (int) inputBarWidth, (int) inputBarHeight,
                          5, 5);
    */

    // get the number of partition based on the total nodes
    // eg: 3 partitions of the input bar hold 3 input nodes in the
middle
    int partition = totalNodes;
    int partitionWidth = (int) (inputBarWidth/partition);
    // draw each node at the end of the partition
    int i;
    for (i=1; i<=totalNodes; i++) {
      int nodeX = (int) (inputBarX + partitionWidth*(i-1)  +
partitionWidth/2
                         - (Operator.NODE_RADIUS/2));
      int nodeY = (int) (inputBarY + inputBarHeight);

      Shape inputNode = new Ellipse2D.Double(nodeX,nodeY,
                            Operator.NODE_RADIUS,Operator.NODE_RADIUS);
      g.draw(inputNode);

      // Hold the position for each node
```

```
        InputBarNode node = (InputBarNode) getNode(i);
        if (node != null) {
          node.ox = nodeX;
          node.oy = nodeY;
        }

        // draw the number
        // Just draw the number
        g.drawString(""+i,nodeX, nodeY-2);

        /*
        // relation node
        if (i <= numRelations) {
          g.drawString(""+i,nodeX, nodeY-2);
        }
        int total = (numRelations + numConditions);
        // condition node
        if ((numConditions > 0) && (i>numRelations) && (i<=total)) {
          g.drawString(""+ (i-numRelations), nodeX, nodeY-2);
        }
        // attributes node
        if ((numAttributes > 0) && (i>total)) {
          g.drawString("" + (i-total), nodeX, nodeY-2);
        }
        */
      }

      // draw the border line and name "Relation(s)"
      if (numRelations > 0) {
        int totalWidth = partitionWidth*numRelations;
        int borderX = (int) (inputBarX + totalWidth);
        int borderY = (int) inputBarY - 15;

        // draw the vertical straight line
        g.drawLine(borderX, borderY, borderX, (int) inputBarY);

        g.setFont(new Font(holdFont.getName(),
                           holdFont.getStyle(),
                           holdFont.getSize() - 2));
        fm = g.getFontMetrics();

        double nWidth = fm.stringWidth("Relation(s)");

        // draw name "relation"
        int posX = (int) (inputBarX + (totalWidth - nWidth)/2);

        g.drawString("Relation(s)", posX, (int) (inputBarY - 5));
      }

      // draw the border line and name "Condition(s)"
      if (numConditions > 0) {
        int totalWidth = partitionWidth*numConditions;
        int borderX = (int) (inputBarX + totalWidth +
partitionWidth*numRelations);
        int borderY = (int) inputBarY - 15;

        // draw the vertical straight line
        g.drawLine(borderX, borderY, borderX, (int) inputBarY);
```

```
      g.setFont(new Font(holdFont.getName(),
                         holdFont.getStyle(),
                         holdFont.getSize() - 2));
      fm = g.getFontMetrics();

      double nWidth = fm.stringWidth("Condition(s)");

      // draw name "Condition(s)"
      int posX = (int) (inputBarX +
                        partitionWidth*numRelations +
                        (totalWidth - nWidth)/2);

      g.drawString("Condition(s)", posX, (int) (inputBarY - 5));
    }

    // draw the border line and name "Attribute(s)"
    if (numAttributes > 0) {
      int totalWidth = partitionWidth*numAttributes;
      int borderX = (int) (inputBarX + totalWidth +
                           partitionWidth*(numRelations+numConditions));
      int borderY = (int) inputBarY - 15;

      // draw the vertical straight line
      g.drawLine(borderX, borderY, borderX, (int) inputBarY);

      g.setFont(new Font(holdFont.getName(),
                         holdFont.getStyle(),
                         holdFont.getSize() - 2));
      fm = g.getFontMetrics();

      double nWidth = fm.stringWidth("Attribute(s)");

      // draw name "Attributes(s)"
      int posX = (int) (inputBarX +
                        partitionWidth*(numRelations+numConditions) +
                        (totalWidth - nWidth)/2);

      g.drawString("Attribute(s)", posX, (int) (inputBarY - 5));
    }

   // after all, change the font back to original
   g.setFont(holdFont);


  }

  public Object getNode(int sequence) {
      Object ob = null;
      boolean bFound = false;

      for (Enumeration e=vRefUserOperator.elements();
e.hasMoreElements() && (!bFound) ;) {
        ob = e.nextElement();
        String className = ob.getClass().getName();
        //System.out.println("OpeatorUser->getNode(), Class Name = " +
className);
        if (className.equalsIgnoreCase("InputBarNode")) {
```

```
            bFound = (sequence == (((InputBarNode) ob).sequence));
          }
        }

        if (bFound)
          return ob;

        // if not found, return null
        return null;

    }

    // Determine if the current mouse position is inside the input bar
    public boolean isInsideInputBar(int cx, int cy) {
      if (((cx >= inputBarX) && (cx <= (inputBarX+inputBarWidth))) &&
          ((cy >= inputBarY) && (cy <= (inputBarY+inputBarHeight)))) {
            // System.out.println("OperatorUser->isInsideInputBar(),
return true");
            return true;
      }

      // System.out.println("OperatorUser->isInsideInputBar(), return
false");
      return false;

    }

    // Determine which node is chosen by the mouse click
    // if no node is chosen, return 0
    public int whichNode(int cx, int cy) {
       if (!isInsideInputBar(cx, cy)) {
          return 0;
       }

       // it is inside the input bar
      int partition = totalNodes;
      int partitionWidth = (int) (inputBarWidth/partition);

      int iNode = (int) ((cx - inputBarX)/partitionWidth) + 1;

      // System.out.println("OperatorUser->whichNode(), iNode = " +
iNode);
      return iNode;

    }

    // Get the object from the design collection base the operator name
    // This is very similar routine as getObject()
    // except the Vector object reference to the DESIGN Vector
vRefUserOperator
    //        instead of the INUSED vector vRefDFQLOperators
    public Object getObjectFromDesignCollection(String operatorName) {
        Object ob = null;
        boolean bFound = false;

        for (Enumeration e=vRefUserOperator.elements();
e.hasMoreElements() && (!bFound) ;) {
          ob = e.nextElement();
```

```
        if (ob.getClass().getName().indexOf("Operator") != -1) {
          bFound = (((Operator)
ob).operatorName).equalsIgnoreCase(operatorName);
        }
      }

      if (bFound)
        return ob;

      // if not found, return null
      return null;

  }


  // Draw the links between input node and operator
  public void drawLink(Graphics g) {
      for (int i=1; i<= totalNodes; i++) {
        // Retrieve the input node
        InputBarNode iNode = (InputBarNode) getNode(i);
        // Obtain the operator name
        String operatorName = iNode.targetOperatorName;
        // System.out.println("OperatorUser->drawLink(), operator name =
" + operatorName);
        if (operatorName.length() > 0) {
          // Get the operator object
          Object ob = getObjectFromDesignCollection(operatorName);
          if (ob != null) {
            Operator op = (Operator) ob;

            double linkX = 0; double linkY = 0;

            switch (iNode.targetOperatorNode) {
              case 1:
                linkX = op.x1; linkY = op.y1;
                break;
              case 2:
                linkX = op.x2; linkY = op.y2;
                break;
              case 3:
                linkX = op.x3; linkY = op.y3;
                break;
              case 4:
                linkX = op.x4; linkY = op.y4;
                break;

              default:

            }

            g.drawLine((int) (iNode.ox + Operator.NODE_RADIUS/2),
                       (int) iNode.oy + Operator.NODE_RADIUS,
                       (int) (linkX + Operator.NODE_RADIUS/2),
                       (int) linkY);
          } else {
            System.out.println("OperatorUser->drawLink(), no found - " +
operatorName);
          }
```

```
      }

    }
}

// draw the connections based on the each input nodes
// this method is called when is in INUSED mode
public void drawConnection(Graphics graphics) {
  double linkX = 0;     double linkY = 0;
  double relateX = 0;   double relateY = 0;


  if (graphics == null) {
    return;
  }

  Graphics2D g = (Graphics2D) graphics;

  totalNodes = numRelations + numConditions + numAttributes;
  if ((totalNodes < 1) || (totalNodes >4)) {
    JOptionPane.showMessageDialog(null,
                                  "Too a few or too much nodes",
                                  "Future feature",
                                  JOptionPane.ERROR_MESSAGE);
    return ;

  }

  switch (totalNodes) {
    case 2:
       for (int i=1; i<= 2; i++) {
        switch (i) {
          case 1:
            linkX = this.x1;  linkY = this.y1;
            break;
          case 2:
            linkX = this.x2;  linkY = this.y2;
            break;
         default:
        }

        // Retrieve the input node
        InputBarNode iNode = (InputBarNode) getNode(i);
        if ((iNode.getInputValue().length() > 0) &&
            (iNode.getNodeType() == InputBarNode.RELATION)) {
          // check if this relation is in the collection
          Object ob = getObjectFromDesignCollection(operatorName);
          if (ob == null) {
            // No found
            switch (i) {
              case 1:
                relateX = getX();
                relateY = getY() - getHeight();
                break;
              case 2:
                relateX = getX() + getWidth();
```

```
                    relateY = getY() - getHeight();
                   break;
                  default:
                }


                // draw the node
                Shape relateNode = new Ellipse2D.Double(relateX,
                                                        relateY,

Operator.NODE_RADIUS,NODE_RADIUS);

                g.draw(relateNode);

                // Draw the conection between two nodes
                g.drawLine((int) (linkX + Operator.NODE_RADIUS/2),
                           (int) linkY,
                           (int) (relateX + Operator.NODE_RADIUS/2),
                           (int) (relateY + Operator.NODE_RADIUS));

                // Calculate the length of the name

                Font holdFont = graphics.getFont();
                graphics.setFont(new Font(holdFont.getName(),
                                          holdFont.getStyle(),
                                          holdFont.getSize() - 2));
                FontMetrics fm = graphics.getFontMetrics();

                // System.out.println("Font size = " +
holdFont.getSize());

                double nameWidth = fm.stringWidth(iNode.getInputValue()) +
2;
                double nameHeight = fm.getHeight() + 6;

                // draw a straight line
                g.drawLine((int) (relateX - (nameWidth/2)),(int) relateY,
                           (int) (relateX + (nameWidth/2)),(int) relateY);

                // draw the relation name
                g.drawString(iNode.getInputValue(),
                             (int) (relateX - (nameWidth/2) + 1),
                             (int) (relateY - 3));


               // after all, change the font back to original
               graphics.setFont(holdFont);

              } else {
                // This object is inside the design collection object
                Operator op = (Operator) ob;

                g.drawLine((int) (op.ox + Operator.NODE_RADIUS/2),
                           (int) (op.oy + Operator.NODE_RADIUS),
                           (int) (linkX + Operator.NODE_RADIUS/2),
                           (int) (linkY));

              }
```

```
          } else {
            if (iNode.getInputValue().length() > 0) {
              // System.out.println("Node seq: " + iNode.getSequence() +
              //                    " where inputValue = " +
iNode.getInputValue());
              // draw everything that is not relation type
              switch (i) {
                case 1:
                  relateX = x1 - getWidth()/2;
                  relateY = getY() - getHeight();
                  break;
                case 2:
                  relateX = x2;
                  relateY = y2 - getHeight() * 2;
                 break;
               default:
              }


              // draw the node
              Shape relateNode = new Ellipse2D.Double(relateX,
                                                      relateY,

Operator.NODE_RADIUS,NODE_RADIUS);

              g.draw(relateNode);

              // Draw the conection between two nodes
              g.drawLine((int) (linkX + Operator.NODE_RADIUS/2),
                         (int) linkY,
                         (int) (relateX + Operator.NODE_RADIUS/2),
                         (int) (relateY + Operator.NODE_RADIUS));

              // Calculate the length of the name

              Font holdFont = graphics.getFont();
              graphics.setFont(new Font(holdFont.getName(),
                                        holdFont.getStyle(),
                                        holdFont.getSize() - 2));
              FontMetrics fm = graphics.getFontMetrics();

              // System.out.println("Font size = " +
holdFont.getSize());

              double nameWidth = fm.stringWidth(iNode.getInputValue()) +
2;
              double nameHeight = fm.getHeight() + 6;

              // draw a straight line
              g.drawLine((int) (relateX - (nameWidth/2)),(int) relateY,
                         (int) (relateX + (nameWidth/2)),(int) relateY);

              // draw the relation name
              g.drawString(iNode.getInputValue(),
                           (int) (relateX - (nameWidth/2) + 1),
                           (int) (relateY - 3));
```

```
            // after all, change the font back to original
            graphics.setFont(holdFont);

          }
        }

      }
      break;
    case 3:
      for (int i=1; i<= 3; i++) {
        switch (i) {
          case 1:
            linkX = this.x1;  linkY = this.y1;
            break;
          case 2:
            linkX = this.x2;  linkY = this.y2;
            break;
          case 3:
            linkX = this.x3;  linkY = this.y3;
            break;
         default:
        }

        // Retrieve the input node
        InputBarNode iNode = (InputBarNode) getNode(i);
        if ((iNode.getInputValue().length() > 0) &&
            (iNode.getNodeType() == InputBarNode.RELATION)) {
          // check if this relation is in the collection
          Object ob = getObjectFromDesignCollection(operatorName);
          if (ob == null) {
            // No found
            switch (i) {
              case 1:
                relateX = x1 - getWidth()/2;
                relateY = getY() - getHeight();
                break;
              case 2:
                relateX = x2;
                relateY = y2 - getHeight() * 2;
               break;
              case 3:
                relateX = x3 + getWidth()/2;
                relateY = getY() - getHeight();
                break;
              default:
            }


            // draw the node
            Shape relateNode = new Ellipse2D.Double(relateX,
                                                    relateY,
Operator.NODE_RADIUS,NODE_RADIUS);

            g.draw(relateNode);

            // Draw the conection between two nodes
```

```
                g.drawLine((int) (linkX + Operator.NODE_RADIUS/2),
                           (int) linkY,
                           (int) (relateX + Operator.NODE_RADIUS/2),
                           (int) (relateY + Operator.NODE_RADIUS));

                // Calculate the length of the name

                Font holdFont = graphics.getFont();
                graphics.setFont(new Font(holdFont.getName(),
                                          holdFont.getStyle(),
                                          holdFont.getSize() - 2));
                FontMetrics fm = graphics.getFontMetrics();

                // System.out.println("Font size = " +
holdFont.getSize());

                double nameWidth = fm.stringWidth(iNode.getInputValue()) +
2;
                double nameHeight = fm.getHeight() + 6;

                // draw a straight line
                g.drawLine((int) (relateX - (nameWidth/2)),(int) relateY,
                           (int) (relateX + (nameWidth/2)),(int) relateY);

                // draw the relation name
                g.drawString(iNode.getInputValue(),
                             (int) (relateX - (nameWidth/2) + 1),
                             (int) (relateY - 3));


               // after all, change the font back to original
               graphics.setFont(holdFont);

              } else {
                // This object is inside the design collection object
                Operator op = (Operator) ob;

                g.drawLine((int) (op.ox + Operator.NODE_RADIUS/2),
                           (int) (op.oy + Operator.NODE_RADIUS),
                           (int) (linkX + Operator.NODE_RADIUS/2),
                           (int) (linkY));

              }

            } else {
              if (iNode.getInputValue().length() > 0) {
                // System.out.println("Node seq: " + iNode.getSequence() +
                //                    " where inputValue = " +
iNode.getInputValue());
                // draw everything that is not relation type
                switch (i) {
                  case 1:
                    relateX = x1 - getWidth()/2;
                    relateY = getY() - getHeight();
                    break;
                  case 2:
                    relateX = x2;
                    relateY = y2 - getHeight() * 2;
```

```
                    break;
                  case 3:
                    relateX = x3 + getWidth()/2;
                    relateY = getY() - getHeight();
                    break;
                  default:
                }


                // draw the node
                Shape relateNode = new Ellipse2D.Double(relateX,
                                                        relateY,

Operator.NODE_RADIUS,NODE_RADIUS);

                g.draw(relateNode);

                // Draw the conection between two nodes
                g.drawLine((int) (linkX + Operator.NODE_RADIUS/2),
                           (int) linkY,
                           (int) (relateX + Operator.NODE_RADIUS/2),
                           (int) (relateY + Operator.NODE_RADIUS));

                // Calculate the length of the name

                Font holdFont = graphics.getFont();
                graphics.setFont(new Font(holdFont.getName(),
                                          holdFont.getStyle(),
                                          holdFont.getSize() - 2));
                FontMetrics fm = graphics.getFontMetrics();

                // System.out.println("Font size = " +
holdFont.getSize());

                double nameWidth = fm.stringWidth(iNode.getInputValue()) +
2;
                double nameHeight = fm.getHeight() + 6;

                // draw a straight line
                g.drawLine((int) (relateX - (nameWidth/2)),(int) relateY,
                           (int) (relateX + (nameWidth/2)),(int) relateY);

                // draw the relation name
                g.drawString(iNode.getInputValue(),
                             (int) (relateX - (nameWidth/2) + 1),
                             (int) (relateY - 3));


               // after all, change the font back to original
               graphics.setFont(holdFont);

              }
            }
          }
          break;
        case 4:
          for (int i=1; i<= 4; i++) {
            switch (i) {
```

```
            case 1:
              linkX = this.x1;  linkY = this.y1;
              break;
            case 2:
              linkX = this.x2;  linkY = this.y2;
              break;
            case 3:
              linkX = this.x3;  linkY = this.y3;
              break;
            case 4:
              linkX = this.x4;  linkY = this.y4;
              break;
           default:
          }

          // Retrieve the input node
          InputBarNode iNode = (InputBarNode) getNode(i);
          if ((iNode.getInputValue().length() > 0) &&
              (iNode.getNodeType() == InputBarNode.RELATION)) {
            // check if this relation is in the collection
            Object ob = getObjectFromDesignCollection(operatorName);
            if (ob == null) {
              // No found
              switch (i) {
                case 1:
                  relateX = x1 - getWidth()/2;
                  relateY = getY() - getHeight();
                  break;
                case 2:
                  relateX = x2;
                  relateY = y2 - getHeight() * 2;
                 break;
                case 3:
                  relateX = x3;
                  relateY = getY() - getHeight()*1.5;
                  break;
                case 4:
                  relateX = x4 + getWidth()/2;
                  relateY = getY() - getHeight();
                  break;
                default:
              }


              // draw the node
              Shape relateNode = new Ellipse2D.Double(relateX,
                                                      relateY,

Operator.NODE_RADIUS,NODE_RADIUS);

              g.draw(relateNode);

              // Draw the conection between two nodes
              g.drawLine((int) (linkX + Operator.NODE_RADIUS/2),
                         (int) linkY,
                         (int) (relateX + Operator.NODE_RADIUS/2),
                         (int) (relateY + Operator.NODE_RADIUS));
```

```
                // Calculate the length of the name

                Font holdFont = graphics.getFont();
                graphics.setFont(new Font(holdFont.getName(),
                                          holdFont.getStyle(),
                                          holdFont.getSize() - 2));
                FontMetrics fm = graphics.getFontMetrics();

                // System.out.println("Font size = " +
holdFont.getSize());

                double nameWidth = fm.stringWidth(iNode.getInputValue()) +
2;
                double nameHeight = fm.getHeight() + 6;

                // draw a straight line
                g.drawLine((int) (relateX - (nameWidth/2)),(int) relateY,
                           (int) (relateX + (nameWidth/2)),(int) relateY);

                // draw the relation name
                g.drawString(iNode.getInputValue(),
                             (int) (relateX - (nameWidth/2) + 1),
                             (int) (relateY - 3));


               // after all, change the font back to original
               graphics.setFont(holdFont);

              } else {
                // This object is inside the design collection object
                Operator op = (Operator) ob;

                g.drawLine((int) (op.ox + Operator.NODE_RADIUS/2),
                           (int) (op.oy + Operator.NODE_RADIUS),
                           (int) (linkX + Operator.NODE_RADIUS/2),
                           (int) (linkY));

              }

            } else {
              if (iNode.getInputValue().length() > 0) {
                // System.out.println("Node seq: " + iNode.getSequence() +
                //                    " where inputValue = " +
iNode.getInputValue());
                // draw everything that is not relation type
                switch (i) {
                  case 1:
                    relateX = x1 - getWidth()/2;
                    relateY = getY() - getHeight();
                    break;
                  case 2:
                    relateX = x2;
                    relateY = y2 - getHeight() * 2;;
                   break;
                  case 3:
                    relateX = x3;
                    relateY = getY() - getHeight()*1.5;
                    break;
```

```
                case 4:
                  relateX = x4 + getWidth()/2;
                  relateY = getY() - getHeight();
                  break;
                default:
              }


              // draw the node
              Shape relateNode = new Ellipse2D.Double(relateX,
                                                      relateY,

Operator.NODE_RADIUS,NODE_RADIUS);

              g.draw(relateNode);

              // Draw the conection between two nodes
              g.drawLine((int) (linkX + Operator.NODE_RADIUS/2),
                         (int) linkY,
                         (int) (relateX + Operator.NODE_RADIUS/2),
                         (int) (relateY + Operator.NODE_RADIUS));

              // Calculate the length of the name

              Font holdFont = graphics.getFont();
              graphics.setFont(new Font(holdFont.getName(),
                                        holdFont.getStyle(),
                                        holdFont.getSize() - 2));
              FontMetrics fm = graphics.getFontMetrics();

              // System.out.println("Font size = " +
holdFont.getSize());

              double nameWidth = fm.stringWidth(iNode.getInputValue()) +
2;
              double nameHeight = fm.getHeight() + 6;

              // draw a straight line
              g.drawLine((int) (relateX - (nameWidth/2)),(int) relateY,
                         (int) (relateX + (nameWidth/2)),(int) relateY);

              // draw the relation name
              g.drawString(iNode.getInputValue(),
                           (int) (relateX - (nameWidth/2) + 1),
                           (int) (relateY - 3));


             // after all, change the font back to original
             graphics.setFont(holdFont);

            }
          }
        }
        break;

      default:
    }
```

```
}

public void draw(Graphics g) {
  // System.out.println("OperatorUser->draw()");
  if (g == null) {
    System.out.println("graphics is null on OperatorUser->draw()");
    return;
  }

  if (mode == DFQL.DESIGN) {
    // this is on DESIGN mode

    // Draw the input bar
    drawInputBar(g);

  } else {
    // this is on INUSED mode
    // Draw the main body
    super.draw(g);
    // draw the nodes
    totalNodes = numRelations + numConditions + numAttributes;
    switch (totalNodes) {
      case 1:
        JOptionPane.showMessageDialog(null,
                                      "Are you playing the system?",
                                      "Only one input node???",
                                      JOptionPane.ERROR_MESSAGE);
        break;
      case 2:
        super.drawTwoInputNodes(g);
        break;
      case 3:
        super.drawThreeInputNodes(g);
        break;
      case 4:
        super.drawFourInputNodes(g);
        break;
      default:
        JOptionPane.showMessageDialog(null,
                                      "Future feature",
                                      "Too many nodes",
                                      JOptionPane.ERROR_MESSAGE);

    }

    // draw the connections
    drawConnection(g);
  }

  // now, the operator should not be redrawn again
  super.setDirty(false);

}

// Action area
public String buildQuery() {
  Operator op = null;
```

```
    // First, set the value to all the link node
    for (int i=1; i<=totalNodes; i++) {
      InputBarNode iNode = (InputBarNode) getNode(i);
      // Obtain the operator name
      String operatorName = iNode.targetOperatorName;
      // System.out.println("OperatorUser->buildQuery(), operator name =
" + operatorName);
      if (operatorName.length() > 0) {
        // Get the operator object
        Object ob = getObjectFromDesignCollection(operatorName);
        if (ob != null) {
          op = (Operator) ob;
          op.setInputNodeValue(i, iNode.getInputValue());
        }
      }
    }

    // Second
    // Get the last operator in the User Operator collection
    op = (Operator) vRefUserOperator.lastElement();

    String qry = op.buildQuery();

    // Third, reset the original value back to all the link node
    for (int i=1; i<=totalNodes; i++) {
      InputBarNode iNode = (InputBarNode) getNode(i);
      // Obtain the operator name
      String operatorName = iNode.targetOperatorName;
      // System.out.println("OperatorUser->buildQuery(), operator name =
" + operatorName);
      if (operatorName.length() > 0) {
        // Get the operator object
        Object ob = getObjectFromDesignCollection(operatorName);
        if (ob != null) {
          op = (Operator) ob;
          op.setInputNodeValue(i, "");
        }
      }
    }



    System.out.println("OperatorUser->buildQuery():\n" + qry);

    return qry;

  }

  // Implement Externalizable interface - write
  public void writeExternal(ObjectOutput out) throws IOException {
      out.writeObject(designName);
      out.writeInt(numRelations);
      out.writeInt(numConditions);
      out.writeInt(numConditions);
      out.writeInt(mode);
      // write the vector object
```

```
      // this object contains the input bar nodes and regular operators
      out.writeObject(vRefUserOperator);

    if (mode == DFQL.INUSED) {
    // Call the super class to save the common data
      super.writeExternal(out);
   }
  }


  // Implement Externalizable interface - read
  public void readExternal(ObjectInput in) {
    // read the data belongs to this operator
    try {

      designName = (String) in.readObject();
      numRelations = in.readInt();
      numConditions = in.readInt();
      numAttributes = in.readInt();
      mode = in.readInt();
      vRefUserOperator = (Vector) in.readObject();

      if (mode == DFQL.INUSED) {

        // call the super class to get the common data
        super.readExternal(in);


      }
   } catch (Exception e) {
      System.err.println(e);
   }
  }

}
```

25. PropertyWindow.java

```
/*
 * File: PropertyWindow.java
 * Author: Ron Chen
 *
 * A extended dialog class for display operator's property
 *
*/

import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;


public class PropertyWindow extends Dialog {
```

```
  static String[] options = { "OK", "CANCEL" };
  static String   title = "Property";

  public static final int OK = 1;
  public static final int CANCEL = 2;

  JButton     okButton;
  JButton     cancelButton;

  JPanel      propertyPanel;

  public int  propertyOption;

  public PropertyWindow(JFrame owner) {
    super(owner);
  }

  public PropertyWindow(JFrame owner,
                        JLabel[] labels,
                        JTextField[] textField) {

    this(owner);
        propertyPanel = new JPanel(false);
        propertyPanel.setLayout(new BoxLayout(propertyPanel,
                                    BoxLayout.X_AXIS));

    JPanel namePanel = new JPanel(false);
    namePanel.setLayout(new GridLayout(0, 1));
    JPanel fieldPanel = new JPanel(false);
    fieldPanel.setLayout(new GridLayout(0, 1));

    for (int i=0; i<labels.length; ++i) {
      namePanel.add(labels[i]);
      fieldPanel.add(textField[i]);
    }

    // Place to the propertyPanel
      propertyPanel.add(namePanel);
      propertyPanel.add(fieldPanel);

    int nOption = JOptionPane.showOptionDialog(owner, propertyPanel,
title,

JOptionPane.DEFAULT_OPTION,

JOptionPane.INFORMATION_MESSAGE,
                                                null, options,
options[0]);

    switch (nOption)
    {
      case 0:
        propertyOption = OK;
        break;
      case 1:
        propertyOption = CANCEL;
        break;
      case JOptionPane.CLOSED_OPTION:
```

```
            propertyOption = CANCEL;
            break;
          default:
        }


    }

}
```

26. TableMap.java

```
/*
 * @(#)TableMap.java    1.4 97/12/17
 *
 * Copyright (c) 1997 Sun Microsystems, Inc. All Rights Reserved.
 *
 * This software is the confidential and proprietary information of Sun
 * Microsystems, Inc. ("Confidential Information").  You shall not
 * disclose such Confidential Information and shall use it only in
 * accordance with the terms of the license agreement you entered into
 * with Sun.
 *
 * SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF
THE
 * SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
 * PURPOSE, OR NON-INFRINGEMENT. SUN SHALL NOT BE LIABLE FOR ANY DAMAGES
 * SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING
 * THIS SOFTWARE OR ITS DERIVATIVES.
 *
 */

/**
 * In a chain of data manipulators some behaviour is common. TableMap
 * provides most of this behavour and can be subclassed by filters
 * that only need to override a handful of specific methods. TableMap
 * implements TableModel by routing all requests to its model, and
 * TableModelListener by routing all events to its listeners. Inserting
 * a TableMap which has not been subclassed into a chain of table
filters
 * should have no effect.
 *
 * @version 1.4 12/17/97
 * @author Philip Milne
 *
 * Modified by: Ron Chen
 * Date: Feb 09, 1999
 * JDK 1.2 places Swing set into different location
 *
 *
 */
import javax.swing.table.*;
import javax.swing.event.TableModelListener;
```

```
import javax.swing.event.TableModelEvent;

public class TableMap extends AbstractTableModel implements
TableModelListener
{
    protected TableModel model;

    public TableModel  getModel() {
        return model;
    }

    public void  setModel(TableModel model) {
        this.model = model;
        model.addTableModelListener(this);
    }

    // By default, Implement TableModel by forwarding all messages
    // to the model.

    public Object getValueAt(int aRow, int aColumn) {
        return model.getValueAt(aRow, aColumn);
    }

    public void setValueAt(Object aValue, int aRow, int aColumn) {
        model.setValueAt(aValue, aRow, aColumn);
    }

    public int getRowCount() {
        return (model == null) ? 0 : model.getRowCount();
    }

    public int getColumnCount() {
        return (model == null) ? 0 : model.getColumnCount();
    }

    public String getColumnName(int aColumn) {
        return model.getColumnName(aColumn);
    }

    public Class getColumnClass(int aColumn) {
        return model.getColumnClass(aColumn);
    }

    public boolean isCellEditable(int row, int column) {
         return model.isCellEditable(row, column);
    }
//
// Implementation of the TableModelListener interface,
//

    // By default forward all events to all the listeners.
    public void tableChanged(TableModelEvent e) {
        fireTableChanged(e);
    }
}
```

27. TableSorter.java

```
/*
 * @(#)TableSorter.java 1.5 97/12/17
 *
 * Copyright (c) 1997 Sun Microsystems, Inc. All Rights Reserved.
 *
 * This software is the confidential and proprietary information of Sun
 * Microsystems, Inc. ("Confidential Information").  You shall not
 * disclose such Confidential Information and shall use it only in
 * accordance with the terms of the license agreement you entered into
 * with Sun.
 *
 * SUN MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF
THE
 * SOFTWARE, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
 * PURPOSE, OR NON-INFRINGEMENT. SUN SHALL NOT BE LIABLE FOR ANY DAMAGES
 * SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING
 * THIS SOFTWARE OR ITS DERIVATIVES.
 *
 */

/**
 * A sorter for TableModels. The sorter has a model (conforming to
TableModel)
 * and itself implements TableModel. TableSorter does not store or copy
 * the data in the TableModel, instead it maintains an array of
 * integers which it keeps the same size as the number of rows in its
 * model. When the model changes it notifies the sorter that something
 * has changed eg. "rowsAdded" so that its internal array of integers
 * can be reallocated. As requests are made of the sorter (like
 * getValueAt(row, col) it redirects them to its model via the mapping
 * array. That way the TableSorter appears to hold another copy of the
table
 * with the rows in a different order. The sorting algorthm used is
stable
 * which means that it does not move around rows when its comparison
 * function returns 0 to denote that they are equivalent.
 *
 * @version 1.5 12/17/97
 * @author Philip Milne
 */


import java.util.*;

import javax.swing.JTable;
import javax.swing.table.*;
import javax.swing.table.TableModel;
import javax.swing.event.TableModelEvent;

// Imports for picking up mouse events from the JTable.
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.InputEvent;

public class TableSorter extends TableMap
```

```
{
    int             indexes[];
    Vector          sortingColumns = new Vector();
    boolean         ascending = true;
    int compares;

    public TableSorter()
    {
        indexes = new int[0]; // For consistency.
    }

    public TableSorter(TableModel model)
    {
        setModel(model);
    }

    public void setModel(TableModel model) {
        super.setModel(model);
        reallocateIndexes();
    }

    public int compareRowsByColumn(int row1, int row2, int column)
    {
        Class type = model.getColumnClass(column);
        TableModel data = model;

        // Check for nulls

        Object o1 = data.getValueAt(row1, column);
        Object o2 = data.getValueAt(row2, column);

        // If both values are null return 0
        if (o1 == null && o2 == null) {
            return 0;
        }
        else if (o1 == null) { // Define null less than everything.
            return -1;
        }
        else if (o2 == null) {
            return 1;
        }

/* We copy all returned values from the getValue call in case
an optimised model is reusing one object to return many values.
The Number subclasses in the JDK are immutable and so will not be used
in
this way but other subclasses of Number might want to do this to save
space and avoid unnecessary heap allocation.
*/
        if (type.getSuperclass() == java.lang.Number.class)
            {
                Number n1 = (Number)data.getValueAt(row1, column);
                double d1 = n1.doubleValue();
                Number n2 = (Number)data.getValueAt(row2, column);
                double d2 = n2.doubleValue();

                if (d1 < d2)
                    return -1;
```

```
            else if (d1 > d2)
                return 1;
            else
                return 0;
        }
    else if (type == java.util.Date.class)
        {
            Date d1 = (Date)data.getValueAt(row1, column);
            long n1 = d1.getTime();
            Date d2 = (Date)data.getValueAt(row2, column);
            long n2 = d2.getTime();

            if (n1 < n2)
                return -1;
            else if (n1 > n2)
                return 1;
            else return 0;
        }
    else if (type == String.class)
        {
            String s1 = (String)data.getValueAt(row1, column);
            String s2    = (String)data.getValueAt(row2, column);
            int result = s1.compareTo(s2);

            if (result < 0)
                return -1;
            else if (result > 0)
                return 1;
            else return 0;
        }
    else if (type == Boolean.class)
        {
            Boolean bool1 = (Boolean)data.getValueAt(row1, column);
            boolean b1 = bool1.booleanValue();
            Boolean bool2 = (Boolean)data.getValueAt(row2, column);
            boolean b2 = bool2.booleanValue();

            if (b1 == b2)
                return 0;
            else if (b1) // Define false < true
                return 1;
            else
                return -1;
        }
    else
        {
            Object v1 = data.getValueAt(row1, column);
            String s1 = v1.toString();
            Object v2 = data.getValueAt(row2, column);
            String s2 = v2.toString();
            int result = s1.compareTo(s2);

            if (result < 0)
                return -1;
            else if (result > 0)
                return 1;
            else return 0;
        }
```

```
    }

    public int compare(int row1, int row2)
    {
        compares++;
        for(int level = 0; level < sortingColumns.size(); level++)
            {
                Integer column =
(Integer)sortingColumns.elementAt(level);
                int result = compareRowsByColumn(row1, row2,
column.intValue());
                if (result != 0)
                    return ascending ? result : -result;
            }
        return 0;
    }

    public void  reallocateIndexes()
    {
        int rowCount = model.getRowCount();

        // Set up a new array of indexes with the right number of
elements
        // for the new data model.
        indexes = new int[rowCount];

        // Initialise with the identity mapping.
        for(int row = 0; row < rowCount; row++)
            indexes[row] = row;
    }

    public void tableChanged(TableModelEvent e)
    {
            System.out.println("Sorter: tableChanged");
        reallocateIndexes();

        super.tableChanged(e);
    }

    public void checkModel()
    {
        if (indexes.length != model.getRowCount()) {
            System.err.println("Sorter not informed of a change in
model.");
        }
    }

    public void  sort(Object sender)
    {
        checkModel();

        compares = 0;
        // n2sort();
        // qsort(0, indexes.length-1);
        shuttlesort((int[])indexes.clone(), indexes, 0, indexes.length);
        System.out.println("Compares: "+compares);
    }
```

```
    public void n2sort() {
        for(int i = 0; i < getRowCount(); i++) {
            for(int j = i+1; j < getRowCount(); j++) {
                if (compare(indexes[i], indexes[j]) == -1) {
                    swap(i, j);
                }
            }
        }
    }

    // This is a home-grown implementation which we have not had time
    // to research - it may perform poorly in some circumstances. It
    // requires twice the space of an in-place algorithm and makes
    // NlogN assigments shuttling the values between the two
    // arrays. The number of compares appears to vary between N-1 and
    // NlogN depending on the initial order but the main reason for
    // using it here is that, unlike qsort, it is stable.
    public void shuttlesort(int from[], int to[], int low, int high) {
        if (high - low < 2) {
            return;
        }
        int middle = (low + high)/2;
        shuttlesort(to, from, low, middle);
        shuttlesort(to, from, middle, high);

        int p = low;
        int q = middle;

        /* This is an optional short-cut; at each recursive call,
        check to see if the elements in this subset are already
        ordered.  If so, no further comparisons are needed; the
        sub-array can just be copied.  The array must be copied rather
        than assigned otherwise sister calls in the recursion might
        get out of sinc.  When the number of elements is three they
        are partitioned so that the first set, [low, mid), has one
        element and and the second, [mid, high), has two. We skip the
        optimisation when the number of elements is three or less as
        the first compare in the normal merge will produce the same
        sequence of steps. This optimisation seems to be worthwhile
        for partially ordered lists but some analysis is needed to
        find out how the performance drops to Nlog(N) as the initial
        order diminishes - it may drop very quickly.  */

        if (high - low >= 4 && compare(from[middle-1], from[middle]) <=
0) {
            for (int i = low; i < high; i++) {
                to[i] = from[i];
            }
            return;
        }

        // A normal merge.

        for(int i = low; i < high; i++) {
            if (q >= high || (p < middle && compare(from[p], from[q]) <=
0)) {
                to[i] = from[p++];
            }
```

```
            else {
                to[i] = from[q++];
            }
        }
    }

    public void swap(int i, int j) {
        int tmp = indexes[i];
        indexes[i] = indexes[j];
        indexes[j] = tmp;
    }

    // The mapping only affects the contents of the data rows.
    // Pass all requests to these rows through the mapping array:
"indexes".

    public Object getValueAt(int aRow, int aColumn)
    {
        checkModel();
        return model.getValueAt(indexes[aRow], aColumn);
    }

    public void setValueAt(Object aValue, int aRow, int aColumn)
    {
        checkModel();
        model.setValueAt(aValue, indexes[aRow], aColumn);
    }

    public void sortByColumn(int column) {
        sortByColumn(column, true);
    }

    public void sortByColumn(int column, boolean ascending) {
        this.ascending = ascending;
        sortingColumns.removeAllElements();
        sortingColumns.addElement(new Integer(column));
        sort(this);
        super.tableChanged(new TableModelEvent(this));
    }

    // There is no-where else to put this.
    // Add a mouse listener to the Table to trigger a table sort
    // when a column heading is clicked in the JTable.
    public void addMouseListenerToHeaderInTable(JTable table) {
        final TableSorter sorter = this;
        final JTable tableView = table;
        tableView.setColumnSelectionAllowed(false);
        MouseAdapter listMouseListener = new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                TableColumnModel columnModel =
tableView.getColumnModel();
                int viewColumn =
columnModel.getColumnIndexAtX(e.getX());
                int column =
tableView.convertColumnIndexToModel(viewColumn);
                if(e.getClickCount() == 1 && column != -1) {
                    System.out.println("Sorting ...");
```

```
                    int shiftPressed =
e.getModifiers()&InputEvent.SHIFT_MASK;
                    boolean ascending = (shiftPressed == 0);
                    sorter.sortByColumn(column, ascending);
                }
            }
        };
        JTableHeader th = tableView.getTableHeader();
        th.addMouseListener(listMouseListener);
    }


}
```

28. ToolTipTree.java

```
/*
 * File: ToolTipTree.java
 * Written by: Ron Chen
 *
 * Last modified: April 14, 1999
 *
 * This is the extended class with the capability to
 * display the tooltip text for the tree node
 *
*/

import javax.swing.*;
import javax.swing.tree.*;
import java.awt.event.*;


public class ToolTipTree extends JTree {


  public ToolTipTree() {
    super();
    // Set the MulitLine ToolTip User Interface class
    createToolTip();
    // trigger tooltps on this object
    this.setToolTipText("");
  }


  ToolTipTree(TreeNode root) {
    super(root);
    // Set the MulitLine ToolTip User Interface class
    createToolTip();
    // trigger tooltps on this object
    this.setToolTipText("");
  }

  public JToolTip createToolTip() {
    MultiLineToolTip tip = new MultiLineToolTip();
```

```
    tip.setComponent(this);
    return tip;
  }
  public String getToolTipText(MouseEvent evt) {
    if (this.getRowForLocation(evt.getX(), evt.getY()) == -1) {
      // not on node yet, no tooltip to display
      return (new String(""));
    }

    String tt = "";
    try {
      TreePath curPath = this.getPathForLocation(evt.getX(),
evt.getY());
      DefaultMutableTreeNode treeNode =
          (DefaultMutableTreeNode)
curPath.getPathComponent(curPath.getPathCount()-1);
        tt = ((TreeNodeName) treeNode.getUserObject()).getToolTip();

    } catch (Exception e) {
      // System.err.println("ToolTipTree->getToolTipText(), error: ");
      // System.err.println(e);
      // if not available, return empty string
      return (new String(""));
    }

    return (new String(tt));
  }
}
```

29. TreeNodeName.java

```
/*
 * File: TreeNodeName.java
 * Written by: Ron Chen
 *
 * Last modified: April 14, 1999
 *
 * This is the class for holding the name of the tree node
 * and tooltip text for this tree node
 *
*/


public class TreeNodeName  {
  private String name = "";
  private String toolTip = "";

  public TreeNodeName() {
    // empty constructer
  }

  public TreeNodeName(String nm) {
    setName(nm);
  }
```

```
  public TreeNodeName(String nm, String tt) {
    setName(nm);
    setToolTip(tt);
  }

  public void setName(String nm) {
    name = new String(nm);
  }

  public String getName() {
    return name;
  }

  public void setToolTip(String tt) {
    toolTip = new String(tt);
  }

  public String getToolTip() {
    return toolTip;
  }


  public String toString() {
    return name;
  }

}
```

30. CompileAll.bat

```
REM written by: Ron Chen
REM Compile most of recent modified class files
REM remark all the non-modified command line for faster compilation


cmd /c "javac LoginDialog.java"
cmd /c "javac AboutBox.java"

cmd /c "javac TreeNodeName.java"
cmd /c "javac MultiLineToolTip.java"
cmd /c "javac ToolTipTree.java"

cmd /c "javac TableMap.java"
cmd /c "javac MyTableModel.java"
cmd /c "javac TableSorter.java"

cmd /c "javac DB.java"

cmd /c "javac ExampleFileFilter.java"

cmd /c "javac -deprecation PropertyWindow.java"

cmd /c "javac -deprecation Operator.java"

Rem Basic Operators
```

```
cmd /c "javac -deprecation OperatorSelect.java"
cmd /c "javac -deprecation OperatorProject.java"
cmd /c "javac -deprecation OperatorJoin.java"
cmd /c "javac -deprecation OperatorUnion.java"
cmd /c "javac -deprecation OperatorDiff.java"
cmd /c "javac -deprecation OperatorGroupcnt.java"

Rem Advance Operators
cmd /c "javac -deprecation OperatorGroupALLsatisfy.java"
cmd /c "javac -deprecation OperatorGroupNsatisfy.java"
cmd /c "javac -deprecation OperatorGroupmax.java"
cmd /c "javac -deprecation OperatorGroupmin.java"
cmd /c "javac -deprecation OperatorGroupavg.java"
cmd /c "javac -deprecation OperatorIntersect.java"

Rem User Defined Operator
cmd /c "javac -deprecation InputBarNode.java"
cmd /c "javac -deprecation OperatorUser.java"


cmd /c "javac -deprecation DFQLCanvas.java"
cmd /c "javac -deprecation DFQL.java"
cmd /c "javac -deprecation FrameMain.java"
```

31. BuildNpsThesis.sql

```
-- Written by: Ron Chen
-- Date: April 5, 1999
-- Name: BuildNpsThesis.sql
-- Description: Build the NPS-CS Thesis database on Oracle database
-- by using this script

-- error handling...
WHENEVER OSERROR CONTINUE;
WHENEVER SQLERROR CONTINUE;

-- log the process
spool BuildNpsThesis.log;

prompt Drop existing user NPSCS
-- connect internal/oracle;
drop user NPSCS cascade;


prompt Create user NPSCS
-- create user name NPS
create user NPSCS identified by npscs
temporary tablespace Temporary_data ;


prompt Grand privileges to NPSCS
-- grant privileges
grant DBA to npscs;

prompt now login as NPSCS
```

```
-- login as user NPSCS
connect npscs/npscs;


-- Save all the changes
commit;


prompt Create table COURSE
-- Create table COURSE
create table COURSE (
  cno          varchar2(10)          not null,
  title        varchar2(20)          not null,
  ino          varchar2(5)           not null,
  CONSTRAINT pk_cno
  PRIMARY KEY (cno)
);

-- commit the change
commit;

prompt Create table ENROLL
-- Create table ENROLL
create table ENROLL (
  cno          varchar2(10)          not null,
  sno          varchar2(10)          not null,
  grade        varchar2(2)           not null,
  testscore    number(3)             default 0,
  CONSTRAINT   pk_cno_sno
  PRIMARY KEY (cno, sno)
);

-- commit the change
commit;

prompt Create table INSTRUCTOR
-- Create table INSTRUCTOR
create table INSTRUCTOR (
  ino          varchar2(10)          not null,
  iname        varchar2(20)          not null,
  pay          number(8,2)           default 0,
  CONSTRAINT   pk_ino
  PRIMARY KEY (ino)
);

-- commit the change
commit;

prompt Create table STUDENT
-- Create table STUDENT
create table STUDENT (
  sno          varchar2(10)          not null,
  sname        varchar2(20)          not null,
  addr         varchar2(20)          not null,
  phone        varchar2(15)          not null,
  gpa          number(4,2)           default 0,
  CONSTRAINT   pk_sno
  PRIMARY KEY (sno)
```

```
);

-- commit the change
commit;

prompt Preload data into table COURSE
-- Preload data into table COURSE
insert into COURSE values ('CS05', 'COURSE #5', 'I1');
insert into COURSE values ('CS10', 'COURSE #10', 'I2');
insert into COURSE values ('CS15', 'COURSE #15', 'I3');
insert into COURSE values ('CS20', 'COURSE #20', 'I2');
insert into COURSE values ('CS25', 'COURSE #25', 'I3');
-- Place one extra row
insert into COURSE values ('CS30', 'COURSE #30', 'I1');

-- commit the change
commit;

prompt Preload data into table ENROLL
-- Preload data into table ENROLL
insert into ENROLL values ('CS10', 'S1', 'A', 92);
insert into ENROLL values ('CS15', 'S1', 'C', 72);
insert into ENROLL values ('CS20', 'S1', 'A', 93);
insert into ENROLL values ('CS05', 'S2', 'A', 98);
insert into ENROLL values ('CS10', 'S2', 'A', 95);
insert into ENROLL values ('CS20', 'S2', 'A', 90);
insert into ENROLL values ('CS05', 'S3', 'B', 85);
insert into ENROLL values ('CS10', 'S3', 'A', 91);
insert into ENROLL values ('CS05', 'S4', 'A', 93);
insert into ENROLL values ('CS15', 'S4', 'B', 83);
insert into ENROLL values ('CS25', 'S4', 'A', 94);
insert into ENROLL values ('CS05', 'S5', 'C', 70);
insert into ENROLL values ('CS15', 'S5', 'B', 82);
insert into ENROLL values ('CS20', 'S5', 'A', 94);

-- commit the change
commit;

prompt Preload data into table INSTRUCTOR
-- Preload data into table INSTRUCTOR
insert into INSTRUCTOR values ('I1', 'INST #1', 100000.00);
insert into INSTRUCTOR values ('I2', 'INST #2', 50000.00);
insert into INSTRUCTOR values ('I3', 'INST #3', 47380.78);

-- commit the change
commit;

prompt Preload data into table STUDENT
-- Preload data into table STUDENT
insert into STUDENT values ('S1', 'STU #1', 'ROOM 1', '111-1111', 3.85);
insert into STUDENT values ('S2', 'STU #2', 'ROOM 1', '111-1111', 3.40);
insert into STUDENT values ('S3', 'STU #3', 'ROOM 3', '333-3333', 3.75);
insert into STUDENT values ('S4', 'STU #4', 'ROOM 3', '444-4444', 2.85);
insert into STUDENT values ('S5', 'STU #5', 'ROOM 5', '555-5555', 3.30);

-- commit the change
commit;
```

```
-- turn off the spooling
spool off;

-- Exit sqlplus
exit;
```

32. BuildNpsThesis.bat

```
REM ============================================================
REM Written by: Ron Chen
REM Date: April 5, 1999
REM Description: Create a NPS Thesis user
REM Make sure this file is run under the working directory
REM ============================================================
sqlplus system/manager @BuildNpsThesis.sql
```

LIST OF REFERENCES

1. Elmasri, Ramez, Navathe, Shamkant B., *Fundamentals of Database Systems*, The Benjamin/Cummings Publishing Company, Inc., 1989.

2. Clark, Gard J., Wu, Thomas C., *DFQL: Dataflow query language for relational databases*, Information & Management 27 (1994) 1-15.

3. Jaworski, Jamie, *Java 1.2 Unleashed,* Sams, 1998, (**http://www.samspublishing.com**).

4. Eckel, Bruce, *Thanking in Java;* Prentice Hall, 1998, (**http://www.phptr.com**).

5. Schneider, Jeff, Arora, Rajeev, *Usng Enterprise Java*, QUE, 1997, (**http://www.quecorp.com/enterprise/**).

6. JFC – Developer Support. (**http://java.sun.com/products/jfc/support.html**).

7. Java Developer Connection. . (**http://developer.java.sun.com/servlet/SessionServlet?action=showLogin&url=/developer/**).

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center.....2
   8725 John J. Kingman Road, Ste 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Dr.D.C.Bogar.....4
   Chairman
   Department of Computer Science
   Code CS
   Naval Postgraduate School
   Monterey, CA 93943-5100

4. Dr. C. Thomas Wu.....1
   Department of Computer Science
   Code CS/WG
   Naval Postgraduate School
   Monterey, CA 93943-5100

5. LCDR Christopher S. Eagle.....1
   Department of Computer Science
   Code CS/CE
   Naval Postgraduate School
   Monterey, CA 93943-5100

6. Ron Z. Chen.....1
   Defense Manpower Data Center
   400 Gigling Rd
   Seaside, CA 93955